Universidad Carlos III de Madrid
www.uc3m.es

Trabajo Fin de Grado:

SISTEMA DE IDENTIFICACIÓN BIOMÉTRICA BASADO EN HUELLA DACTILAR MEDIANTE BINARIZACIÓN SOBRE PLATAFORMAS ANDROID

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

AUTOR: José Ramón González Isabel

TUTOR: Raúl Sánchez Reíllo

Leganés, 4 de Febrero de 2013

Contenido

Tabla de ilustraciones

Resumen

La constante evolución de la tecnología no supone otra cosa que una creciente dependencia por parte de las personas de todo tipo de dispositivos electrónicos, llegando al punto actual en el que se confía a éstos datos vitales como claves de cuentas bancarias, números de identificación personal o contraseñas de acceso a documentos y archivos personales. Es por esto que surge la necesidad de mantener la privacidad del usuario a toda costa y en todo momento mediante técnicas de seguridad.

La identificación biométrica por huella dactilar está totalmente presente en nuestra vida cotidiana y ha demostrado ser un método eficaz de protección de datos debido a la prácticamente nula similitud entre las huellas dactilares de todas las personas.

Este proyecto intenta unir la última tecnología con este método de seguridad biométrica implementando el sistema de reconocimiento de huella por algoritmo Bozorth en plataformas Android, actualmente muy extendidas, y facilitando una mayor privacidad de sus datos a los usuarios.

Abstract

The constant evolution of technology has made people completely dependent on electronic devices, getting to the point on what people entrust them with sensible data like bank account's keys, identification numbers or private files' passwords. Therefore, there is an extreme need of maintaining the user's privacy at all costs using security techniques.

Fingerprint biometric identification is present in our daily lives and has proven to be such an efficient data protection method, because of the almost null similitude between any two persons.

This project aims to merge the technology with this identification system implementing the Bozorth algorithm based fingerprint identification system for Android devices trying to give full privacy to user's data.

Diccionario

Activity: en Android, cada una de las ventanas que muestra un interfaz de usuario y definida a su vez en la clase View.

Alert Dialog: ventana emergente que proporciona al usuario dos botones con dos opciones diferentes a implementar, así como un título y una descripción de las dos funciones.

App: Aplicación de software.

Array: (Término informático) zona de almacenamiento continuo, que contiene una serie de elementos del mismo tipo, los elementos de la matriz.

ArrayAdapter: adaptador de objetos básicos que controla un array de objetos arbitrarios.

Arraylist: (Término informático) Implementación de lista respaldado por un array. Todas las operaciones opcionales, incluyendo agregar, quitar y remplazar elementos son compatibles.

Bitmap: Mapa de bits, también conocida como imagen matricial, es una estructura o fichero de datos que representa una rejilla rectangular de píxeles opuntos de color, denominada matriz, que se puede visualizar en papel, monitor u otro dispositivo de representación.

BOZORTH3: algoritmo empleado en este proyecto para realizar la comparación entre dos huellas dactilares y generar un resultado numérico.

Bundle: (Término informático) Clase que permite contener tipos primitivos y objetos de otras cases. Es posible mediante esta clase pasar datos entre activities.

Canvas: (Término informático) Clase que mantiene la función de Draw (dibujar). Para usarlo es necesario un bitmap que contenga los píxeles de destino, un bitmap de origen y la pintura o matiz de color a aplicar.

Cast: Término informático referido al cambio de tipo de una variable.

ERR: del inglés *equal error rate*. En las gráficas de distribución normal del falso rechazo y la falsa aceptación, es el punto en el que ambas se cruzan y tienen la misma probabilidad de coexistir.

FAR: del inglés *false acceptance rate (tasa de falsa aceptación)*. Es la probabilidad de que el sistema verifique de forma positiva a un usuario al que la huella que se está procesando no corresponde.

FRR: del inglés *false reject rate (tasa de falso rechazo)*. Es la probabilidad de que el sistema verifique de forma negativa a un usuario al que la huella que se está procesando sí corresponde.

File path: ruta en la que un archivo concreto se encuentra alojado en la memoria.

FprintApp: nombre de la aplicación para dispositivos Android realizada en el presente proyecto.

Intent*:* en Android, clase que permite enviar peticiones para realizar acciones determinadas,

como por ejemplo, la apertura de una activity concreta.

Layout: distribución de los elementos que conforman una ventana a la hora de desarrollar el apartado gráfico de cada una de las activities de la aplicación. Se implementa en los archivos .xml.

Listener: (Término informático) función que implementa la acción que se realizará tras producirse un evento determinado como por ejemplo, la pulsación de un botón.

ListView: (Término informático) elemento de la clase View de Android que presenta una lista dinámica de elementos con interfaz de usuario y posibilidad de desplazamiento vertical.

MINDTCT: algoritmo usado en este proyecto para realizar la detección de minucias de una huella dactilar partiendo de la imagen digitalizada de la misma.

Null: (Término informático) Referencia a la nada aplicable a variables o punteros. Valor que indica que la variable no contiene nada o el puntero apunta a la nada.

String: (Término informático) Secuencia inmutable de caracteres representados como array.

Toast: (Término informático) Vista que contiene un fugaz y pequeño mensaje emergente para el usuario. Esta clase es usada para notificar acciones momentáneamente.

Uri: (Término informático) Clase necesaria para obtener datos de otras actividades o procesos específicos. Esta clase es muy indulgente, devolverá basura (de datos, es decir, datos inservibles) antes de lanzar una excepción a no ser que se le especifique lo contrario.

View: en Android, clase que representa la construcción básica de bloques de componentes que proporcionan interfaz de usuario. Un View ocupa un área rectangular en la pantalla y es responsable del dibujado y de manejar eventos. Ésta es la clase base de los *widgets*, usados para crear componentes interactivos como botones o campos de texto.

Widgets: Pequeña aplicación o programa, usualmente presentado en archivos o ficheros pequeños que son ejecutados por un motor (Widget engine).

XML: lenguaje de marcas que permite definir la gramática de lenguajes específicos. En Android, los archivos xml recogen el código que implementa los elementos visuales de la aplicación, los cuales proporcionan la interfaz de usuario.

.XYT: extensión del archivo de texto que contiene la información de todas las minucias detectadas tras la aplicación del algoritmo MINDTCT.

1. Introducción

A continuación se intentará explicar de manera general el contenido del presente documento.

En esta memoria contiene todos los detalles de un Trabajo de Fin de Grado en el que se desarrolla una aplicación para dispositivos Android basada en el reconocimiento de huella dactilar por binarización mediante algoritmo Bozorth3. El procedimiento se basa en generar un archivo que contendrá las minucias detectadas a partir de una imagen virgen de una huella dactilar. Dicho archivo se usará para compararlo con otras huellas.

El desarrollo de la aplicación se basa en el algoritmo Bozorth3 y los métodos de obtención de la imagen ya implementados anteriormente en lenguaje C#, por lo que durante este documento se presentarán todos los detalles tanto de la conversión del código de C# a Java como de la realización de nuevas funciones incluidas en la aplicación para Android (reclutamiento de usuarios, asignación de huellas a éstos o verificación) y del entorno gráfico que sustenta dicha aplicación.

1.1 Motivación

La principal razón para escoger este Trabajo de Fin de Grado fue la familiarización con el lenguaje de programación Java, actualmente en auge ya que es éste el lenguaje usado para el desarrollo de aplicaciones Android. Se trata de un lenguaje potente con el que se pueden desarrollar aplicaciones con cualquier tipo de funcionalidad que pueda otorgar el dispositivo para el que se programa.

Un aspecto muy destacable es la enorme facilidad encontrada para el desarrollo de aplicaciones Android, ya que Google pone a disposición pública todo tipo de ejemplos de código libre e información a través de los cuales podemos acceder fácilmente a todas las funciones de nuestro dispositivo Android gracias a bibliotecas y funciones predefinidas muy intuitivas y sencillas de implementar.

Otra de las motivaciones para realizar este trabajo ha sido el enorme grado de integración en la sociedad que actualmente poseen las plataformas móviles Android como “smartphones” y “tablets”, las cuáles han causado tal impacto en las personas que en muchas ocasiones han provocado incluso dependencia de las numerosas posibilidades que ofrecen estos dispositivos.

Debido a esto, estos dispositivos se han convertido en pequeñas cajas fuertes, ya que muchas personas les confían datos cruciales de sus vidas como cuentas bancarias, contraseñas de acceso a correos personales o cuentas en distintas Webs. Es por ello que surge la necesidad de buscar métodos de seguridad y protección de datos, dando lugar a otra de las principales motivaciones de este trabajo, la de implementar un sistema de identificación que se ha demostrado muy seguro durante todo este tiempo, llegando a utilizarse en organismos como el F.B.I. para la identificación de personas, en las populares plataformas Android.

Por todo lo anterior, se ha tratado de realizar un sistema seguro, accesible para todo tipo de usuarios y de alta comodidad.

1.2 Objetivos

Los principales objetivos de este trabajo son el de buscar la correcta implementación de este sistema de identificación biométrica en dispositivos basados en el sistema operativo Android y hacerlo de una forma totalmente intuitiva y fácil de utilizar para que pueda ser accesible a todo tipo de público. Además esta aplicación debe ser compatible con cualquier dispositivo Android, evitando que presente errores a la hora de su instalación o ejecución.

Otro objetivo es el de comparar los resultados obtenidos con la versión del algoritmo realizada para el lenguaje C#, pudiendo así establecer diferencias de tiempos de ejecución y generación de errores entre las dos versiones.

2. Introducción a la biometría

A continuación se hablará de los conceptos generales de la biometría y la identificación biométrica aplicada a la seguridad. También se harán apuntes acerca de las distintas técnicas de identificación biométrica, concretando sobre todo en la identificación por huella dactilar.

2.1 Conceptos generales

La biometría es el estudio de métodos de reconocimiento de humanos basados en la extracción de alguna de sus características intrínsecas, ya sean físicas (huella, iris, venas de la mano, etc) o de comportamiento (la firma, el paso, el tecleo, etc).

Este concepto se extrapola a la tecnología de la información como la "autentificación biométrica", que se trata de la aplicación de algoritmos matemáticos y estadísticos para automatizar el reconocimiento de los rasgos físicos o de conducta de un individuo para así poder autentificarlo, es decir, verificar su identidad.

Estos sistemas de reconocimiento se basan en *indicadores biométricos*, que son esas características de las que se hablaba y que permiten realizar el reconocimiento biométrico. Cualquiera que sea ese indicador, debe cumplir los siguientes requisitos:

- *Universalidad:* la característica que se use para el reconocimiento debe ser común a todas las personas.
- *Unicidad:* la probabilidad de que haya dos personas con la misma característica biométrica es muy pequeña.
- *Permanencia:* la característica en cuestión debe mantenerse con el paso del tiempo.
- *Cuantificación:* la característica debe poder medirse de forma cuantitativa.

Estos requisitos que se han citado son los que sirven para descartar o aprobar una característica como *indicador biométrico*. Pero una vez seleccionado este indicador, es necesario establecer una serie de restricciones al sistema que tendrá como misión procesarlo. A continuación se presentan esas restricciones:

- *Rendimiento y eficacia:* el objetivo de esta restricción es comprobar si el sistema presenta la rapidez, robustez y exactitud necesarias para que la identificación sea aceptable. Además, el procesamiento debe poseer un requerimiento de recursos razonable.

- *Aceptabilidad:* indica la medida en que la gente estaría dispuesta a incorporar el sistema de identificación en su vida diaria. Naturalmente, esta restricción descarta todo tipo de sistemas que puedan resultar dañinos o nocivos.

- *Fiabilidad:* esta restricción hace referencia a lo robusto que es el sistema frente a amenazas o intrusiones. El sistema de reconocer características de una persona viva, ya que es posible crear huellas de látex, prótesis de ojos, grabaciones de voz, etc… Algunos de los métodos más ingeniosos que se emplean para el reconocimiento de características vivas son la detección de patrones característicos en las manchas del iris, captación de estructuras subcutáneas mediante ultrasonidos en el reconocimiento de huella dactilar…

2.2 Historia

La biometría no se puso en práctica en las culturas occidentales hasta finales del siglo XIX, sin embargo, hay indicios de que era utilizada en China desde al menos el siglo XIV así como testimonios que aseguran que los comerciantes chinos estampaban las huellas de las palmas de la mano de los niños sobre papel con tinta como método para diferenciar entre los niños jóvenes.

En occidente, el primer sistema de reconocimiento biométrico no llegó hasta el año 1883, cuando el departamento fotográfico de policía de París desarrolló el primer sistema preciso que fue ampliamente usado y dio paso al estudio exhaustivo de la biometría. Este sistema se basaba en medir la anchura y la longitud de cabeza y cuerpo y registrar rasgos característicos como tatuajes o cicatrices. A este sistema se le bautizó con el nombre de método de Bertillón, en honor a su inventor.

Tras aparecer defectos en el sistema como el uso de distintos métodos de medida o la variación de las medidas registradas, se pasó a usar la huella dactilar basándose en el sistema desarrollado por los chinos en el siglo XIV.

Actualmente existen muchas otras técnicas de identificación biométrica como el reconocimiento de iris, de las venas de la mano, del rostro, de la voz, etc. Sin embargo, su amplio reconocimiento como una técnica valida por los sistemas legales de diversos países y su gran experiencia y madurez, han hecho que la identificación biométrica por huella dactilar sea la técnica más ampliamente utilizada en el mundo.

Antiguamente, tanto el volumen como el coste de los sistemas de captura de las huellas, así como su coste computacional, hizo que su aplicación masiva no fuera posible. Sin embargo, la actual evolución tecnológica ha abaratado los costes de los dispositivos utilizados y ha disminuido los recursos necesarios para el procesado, por lo que su implantación global se ve cada vez más real.

2.3 Funcionamiento y técnicas

2.3.1 Funcionamiento

Los sistemas de identificación biométrica poseen tres componentes: el primero de ellos se encarga de la adquisición de algún indicador biométrico de una persona, como por ejemplo, adquirir mediante un escáner la imagen de una huella dactilar. El segundo componente se encarga de la compresión de la información, el procesamiento mediante la aplicación de los algoritmos necesarios, el almacenamiento y la comparación de los datos adquiridos. El tercer componente es la interfaz que facilita el manejo del sistema, la cuál puede encontrarse en el propio sistema o en otro diferente.

Estos tres componentes pueden englobarse en dos módulos que son comunes a todos los sistemas de identificación: el módulo de reclutamiento (enrolment module) y el módulo de identificación (identification module).

Tanto el módulo de identificación como el de verificación son similares, su diferencia radica en que en el de identificación se presenta un indicador biométrico, en nuestro caso la imagen de una huella, y se compara con todos los usuarios de la base de datos para identificar a su dueño. El módulo de verificación, sin embargo, introduce la imagen de la huella para compararla con un usuario en concreto y ver si la identificación es positiva o negativa.

La siguiente figura muestra un esquema del funcionamiento básico de cualquier sistema de identificación biométrica, ejemplificándolo con el caso de que el indicador biométrico se trate de una huella dactilar:

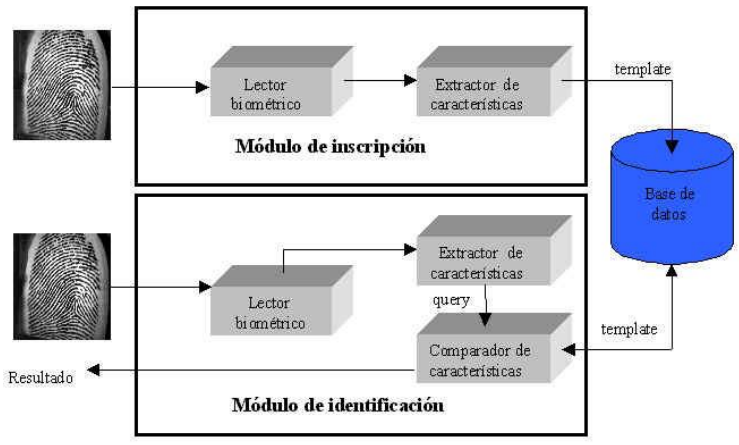

Ilustración 1: Funcionamiento básico de un sistema de identificación [10]

Como se puede observar, no se incluye en la figura el módulo de verificación puesto que su esquema sería idéntico al de identificación.

Ambos módulos comienzan con un lector biométrico que, en nuestro caso, captura la imagen de la huella del individuo y la convierte a formato digital. A continuación, el módulo de

inscripción aplica los algoritmos necesarios para la extracción de sus características y las almacena en una base de datos.

El módulo de identificación, tras la extracción de características, compara éstas con las que ya se encuentran en la base de datos para identificar el usuario al que corresponden o, en el caso de la verificación, para comprobar que el usuario que está intentando acceder al sistema es realmente él.

2.3.2 Técnicas biométricas

La actual evolución tecnológica supone una reducción de los costes de producción de sistemas electrónicos y, por tanto de los dispositivos utilizados para la biometría. También se disminuyen los recursos utilizados por estos dispositivos. En la siguiente figura podemos observar un dispositivo de lectura de huella dactilar. El individuo se encuentra realizando una captura del dedo índice. Esta imagen quedará registrada digitalmente para procesarla de la forma que se desee.

Ilustración 2: Lector de huella dactilar [19]

Todo ello sumado con el constante y creciente estudio de las posibilidades de identificación biométrica hace que surjan diversas técnicas. Estas técnicas de identificación están basadas en los dos tipos de características:

- *Estáticas:* son aquellas características físicas de las personas. Por ello son las más fiables puesto que apenas se modifican con el tiempo y suelen poseer una gran unicidad. Entre estas características se encuentran por ejemplo las huellas dactilares, el iris, la retina, las venas de la mano, el rostro…

- *Dinámicas:* son aquéllas que van asociadas al comportamiento de las personas y por tanto están mucho mas sujetas a cambios. Son ejemplos de estas características la firma, el paso, el tecleo… Estas características son menos fiables que las estáticas.

A continuación se presenta una tabla que recoge las técnicas de identificación más comunes y las compara entre ellas según su fiabilidad, facilidad de uso, robustez, aceptación y estabilidad:

Ilustración 3: Tabla comparativa de técnicas de identificación biométrica [7]

	Ojo (Iris)	Ojo (Retina)	Huellas dactilares
Fiabilidad	Muy alta	Muy Alta	Muy Alta
Facilidad de uso	Media	Baja	Alta
Prevención de ataques	Muy alta	Muy Alta	Alta
Aceptación	Media	Baja	Alta
Estabilidad	Alta	Alta	Alta

Vascular dedo	Vascular mano	Geometría de la mano	Escritura y firma	Voz	Cara 2D	Cara 3D
Muy Alta	Muy Alta	Alta	Media	Alta	Media	Alta
Muy Alta	Muy Alta	Alta	Alta	Alta	Alta	Alta
Muy Alta	Muy Alta	Alta	Media	Media	Media	Alta
Alta	Alta	Alta	Muy Alta	Alta	Muy alta	Muy alta
Alta	Alta	Media	Baja	Media	Media	Alta

Como se puede observar, el reconocimiento de huella dactilar es una técnica muy versátil ya que posee una gran aceptación y facilidad de uso. Además constituye un sistema robusto frente a ataques, ya que si se intenta hackear, el atacante obtendrá una cantidad de números que no sabrá a qué corresponde cada uno. Asimismo es un sistema con gran estabilidad frente a cambios y de alta fiabilidad en los procesos de verificación e identificación.

3. Reconocimiento de huella dactilar

En este apartado trataremos más a fondo el tema que concierne a este proyecto, la técnica de identificación biométrica por reconocimiento de huella dactilar mediante binarización, basada en algoritmo de comparación Bozorth.

3.1 Descripción de la huella dactilar

El reconocimiento de huella dactilar es la técnica más ampliamente utilizada en el mundo en la actualidad. Como hemos explicado anteriormente, los primeros estudios de la huella dactilar se remontan a siglos pasados, sin embargo, los fundamentos de los métodos de identificación por huella que hoy se consideran asentados no se establecería hasta finales del siglo XIX, cuando Sir Edward Henry y Sir Francis Galton trabajaron en el estudio, de forma análoga pero separada, del uso de la huella dactilar como método para identificar personas por clasificación.

En estos estudios se elaboró la descripción de la huella dactilar básicamente como una sucesión de *crestas* separadas entre sí por *valles*. La disposición de estas *crestas* se traduce en la aparición de ciertos puntos singulares denominados *terminaciones* y *bifurcaciones*. Este grupo de puntos singulares recibió el nombre de *minucias* y, mediante su orientación, localización y tipo se puede identificar a una persona. Ciertos autores consideran que hay hasta ocho tipos diferentes de minucias, sin embargo, todos ellos son combinaciones de estas terminaciones y bifurcaciones.

Estudios realizados a finales del siglo XIX revelaron que la estructura de la huella dactilar no pertenece a las capas más superficiales de la piel, si no que se trata de algo intrínseco. Por ello, si por cualquier circunstancia se pierde la piel de un dedo, el recrecimiento de la piel provocará que la huella se vuelva a reconstruir tal y como era antes. Además se ha comprobado que cada huella posee un elevado grado de unicidad, llegando al punto de que un mismo individuo tiene huellas totalmente diferentes en cada uno de sus dedos.

Sin embargo, no solo las minucias sirven para realizar la identificación de un individuo. Existen otros puntos singulares como los llamados *núcleo (core), y delta*, que son muy utilizados para implementar sistemas de reconocimiento. El núcleo podría describirse como el punto en el que la orientación de las crestas tiende a converger, mientras que los deltas serían los puntos donde el flujo de crestas presenta una divergencia.

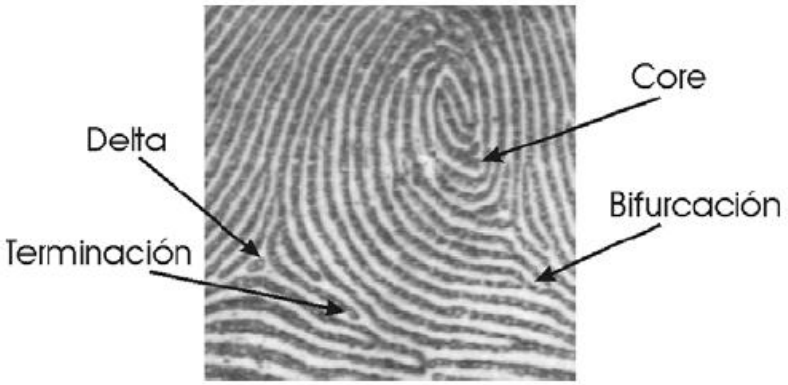

Ilustración 4: Puntos singulares de una huella dactilar [15]

Como podemos observar en la imagen, las terminaciones son los puntos en los que las crestas presentan un extremo marcado, mientras que las bifurcaciones son puntos en los que una cresta se divide en varias.

El sistema de identificación por huella dactilar ha tomado tal importancia que actualmente se aplica en la mayoría de sistemas policiales del mundo, creándose grandes bases de datos para almacenar los datos de todo tipo de personas.

3.2 Algoritmos implementados

A continuación se describirán los algoritmos utilizados para poder desarrollar la aplicación. Estos algoritmos son los que permiten que se pueda decodificar una imagen en escala de grises tomada de un lector, extraer las minucias pertinentes para realizar la identificación y, finalmente, comparar dos archivos de salida con las minucias de cada huella y establecer un resultado numérico para evaluar el parecido entre ellas.

3.2.1 MINDTCT

MINDTCT es un detector de minucias desarrollado por el departamento de imagen del NIST (National Institute of Standards and Technology). Este algoritmo fue desarrollado para el F.B.I. (Federal Bureau of Investigation) con el fin de facilitar el procesado automático y de imágenes de huellas dactilares.

Este algoritmo analiza un archivo compatible con el estándar ANSI/NIST-ITL 1-2000, el cuál se trata de un formato de datos específico para el intercambio de información basada en minucias, concretamente huellas dactilares, información facial y de cicatrices, marcas y tatuajes (SMT).

El algoritmo busca en este archivo la primera aparición del registro de una imagen de huella en escala de grises. Si la encuentra, la imagen es decodificada y procesada, y las minucias

se detectan automáticamente. El algoritmo genera una serie de ficheros como resultado que se enumerarán rápidamente a continuación para proceder más adelante a un análisis más profundo de los mismos:

- *Direction map (mapa de dirección):* representa la dirección del flujo de crestas en la imagen.
- *High curvature map(mapa de alta curvatura):* representa las zonas de la imagen que presentan un flujo de crestas con una curvatura muy pronunciada.
- *Low contrast map (mapa de bajo contraste):* en él se incluyen las áreas de la imagen que presentan zonas con el contraste bajo. A menudo se trata de las zonas de la imagen en las que se encuentra el fondo.
- *Low flow map (mapa de bajo flujo):* contiene las zonas de la imagen en las que no se puede determinar que exista flujo de crestas.
- *Quality map (mapa de calidad):* se trata de un mapa elaborado con la combinación de los demás, el cual contiene valores de 0 a 4 que representan la calidad de cada grupo de píxeles de la imagen.
- *Archivo ".xyt":* es el archivo final, se trata de un archivo de texto al que, en la versión del algoritmo de este proyecto, se le adjudica la extensión ".xyt" que será la que posteriormente utilice como entrada el algoritmo comparador. En este archivo, cada línea de texto no vacía corresponde a una minucia y a los atributos que van asociados a ella.

En secciones más adelante se describe el funcionamiento de este algoritmo de forma más exhaustiva.

3.2.2 Bozorth3

Se trata del algoritmo usado para la comparación de huellas dactilares. Fue escrito por Allan Bozorth en el F.B.I., sin embargo, posteriormente el NIST lo adoptó para mejorarlo. Este tipo de algoritmo puede usarse para comparar huellas para sendos propósitos, identificación ("uno a muchos") o verificación ("uno a uno") y recibe el nombre de *matcher*.

Para entender el funcionamiento del algoritmo es importante destacar que las características de las minucias están exclusivamente limitadas a posición y orientación, representadas como {x, y, t} respectivamente. El algoritmo está diseñado para ser invariante a rotación y traslación. Se compone mayoritariamente de tres pasos que mencionaremos a continuación:

- *Construcción de las tablas de comparación internas a la huella:* se construye una tabla para la huella tomada y otra para aquélla de la base de datos con la que se comparará.

- *Construcción de la tabla de compatibilidad entre huellas:* compara sendas tablas de comparación y genera una nueva tabla de compatibilidad.

- *Recorrido de la tabla de compatibilidad:* se recorre la tabla y se unen sus entradas en grupos. Se combinan los grupos compatibles y se calcula una puntuación.

3.3 Proceso detallado: MINDTCT

En este apartado se detallará el procedimiento que sigue el algoritmo MINDTCT paso por paso, desde que se toma una imagen hasta que se genera la puntuación durante la comparación.

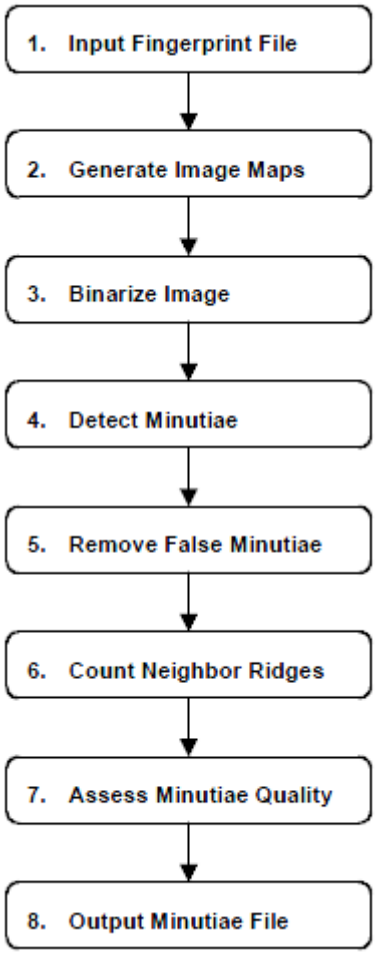

Ilustración 5: Diagrama de flujo de la detección de minucias [15]

En la imagen podemos apreciar un diagrama de flujo de del proceso de detección de la imagen, antes de proceder a la aplicación del algoritmo de comparación Bozorth3.

3.3.1 Imagen de huella de entrada

Como se ha explicado anteriormente, el algoritmo MINDTCT está desarrollado para tomar imágenes cuantificadas en 256 niveles de gris. Al seleccionar una imagen en escala de grises, el algoritmo comienza el procesado de la misma.

3.3.2 Generación de mapas de calidad de imagen

Debido a que la calidad de la imagen puede variar, especialmente en el caso de huellas latentes, es necesario poder analizar la imagen y determinar zonas que pueden estar degradadas y causar problemas. Para analizar las distintas zonas de calidad de la imagen se generan los mapas de los que ya se habló en la descripción global del algoritmo MINDTCT: mapa de dirección, de bajo contraste, de alta curvatura y de bajo flujo.

3.3.2.1 Mapa de dirección

El propósito de este mapa es representar las áreas de la imagen con una estructura de crestas suficiente. Las crestas bien formadas y claramente visibles son necesarias para determinar puntos fiables de detección de fin de cresta y bifurcación. Además, este mapa contiene la orientación general del flujo de crestas a través de la imagen.

Para analizar la huella de forma local, la imagen se divide en una red de bloques. A todos los píxeles dentro de cada bloque se les asigna el mismo resultado, por ello, todos los píxeles de un grupo compartirán la misma dirección de flujo de crestas.

Una vez realizados los grupos, es necesario saber qué cantidad de información local es necesario extraer para derivar de forma fiable la característica deseada. Esta zona es conocida como *ventana (window).* La característica medida en la ventana se asigna a cada píxel del bloque. Es deseable compartir los datos usados para computar los resultados asignados a bloques vecinos. De esta forma, parte de la imagen que contribuye a generar los resultados de un bloque, se incluye también en los resultados de un bloque vecino. Este *suavizado* ayuda a salvar la discontinuidad que se produce al cruzar desde el borde de un bloque al siguiente. Este suavizado se puede implementar usando un sistema en el que un bloque es menor que la *ventana* que lo rodea y, por tanto, esta se solapa de un bloque al siguiente. Se ilustra en la siguiente figura:

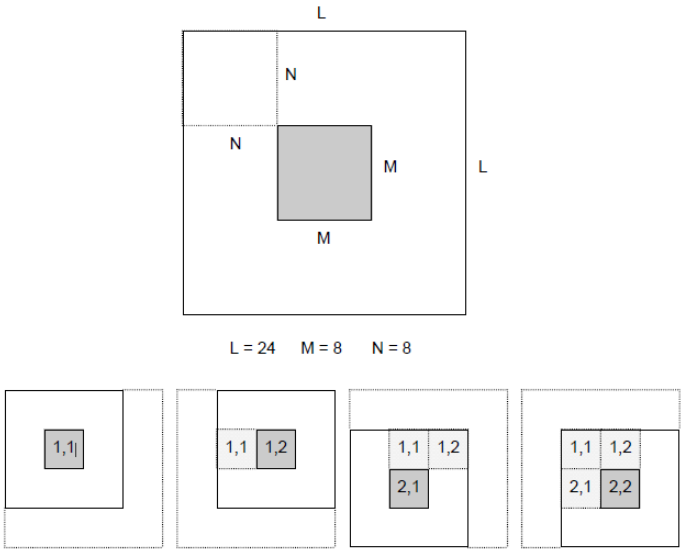

Ilustración 6: Bloques adyacentes con ventanas solapadas [15]

En la figura superior podemos observar una *ventana* (en blanco) que rodea a un bloque (en gris). El esquema se basa en tres parámetros: L (tamaño de la *ventana*), M (tamaño del bloque) y N (offset del bloque desde el origen de la *ventana*). Para entender este esquema asumiremos que los bloques vecinos son adyacentes y no se solapan, por lo tanto comparten información de la misma *ventana.* Esto se puede apreciar en las cuatro imágenes inferiores, en las que se representan bloques designados por su dirección (fila, columna). En el marco de la izquierda se muestra el primer bloque siendo computado por su *ventana* correspondiente. El siguiente marco muestra el bloque adyacente de la derecha y su *ventana* correspondiente, desplazada 8 píxeles a la derecha. Así es como el bloque 1,2 recibe los resultados del 1,1.

Una vez explicado el sistema de *ventanas* solapadas, se describirá la técnica utilizada para determinar la dirección del flujo de crestas. Para cada bloque en la imagen, la *ventana* que lo rodea se rota incrementalmente y un análisis DFT (Discrete Fourier Transform) se realiza para cada orientación. La forma de rotar la ventana es la siguiente:

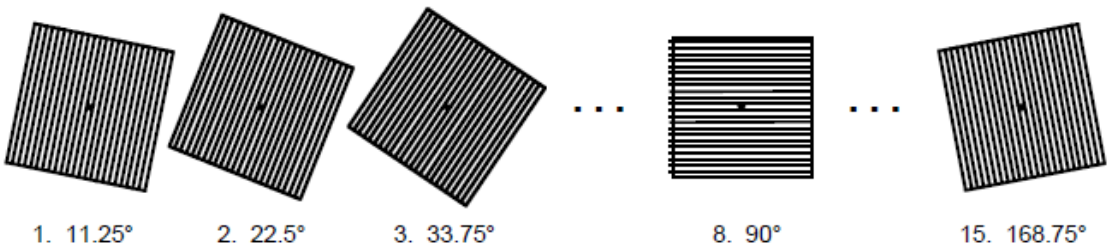

Ilustración 7: Rotación incremental de una ventana para cada orientación [15]

En los parámetros generales del programa está limitado el número de orientaciones por defecto a analizar en un semicírculo a 16. En cada orientación, todos los píxeles de una fila de una *ventana* se suman formando de 24 sumas de fila de píxeles. Las 16 orientaciones producen 16 de estos vectores, los cuáles son representados con formas de onda de frecuencia creciente para distintos valores de seno y coseno como se muestra en la siguiente ilustración:

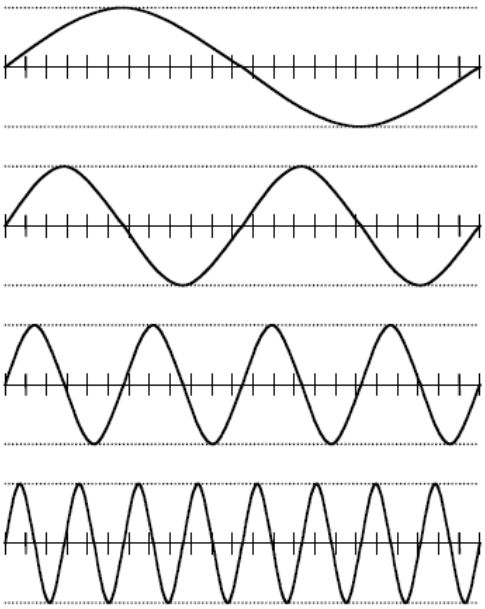

Ilustración 8: Formas de onda con distintas frecuencias [15]

La variación de frecuencia de cada forma de onda es debida a que cada una de ellas representa crestas y valles de distinta anchura. La onda superior representa crestas y valles de aproximadamente 12 píxeles de anchura, la siguiente de 6 píxeles, a continuación de 3 píxeles y, por último, la cuarta forma de onda representa crestas y valles de 1.5 píxeles de anchura.

A continuación se representa, a la izquierda, una imagen de huella original y a la derecha la misma imagen anotada con las distintas orientaciones del flujo de crestas generada por los resultados del mapa de direcciones:

Ilustración 9: Resultados del mapa de direcciones [15]

3.3.2.2 *Mapa de bajo contraste*

Es muy difícil, si no imposible, determinar un flujo de crestas dominante en determinadas zonas de la imagen. Esto se produce en regiones de bajo contraste que contienen fondo de la imagen o manchas. Es necesario detectar estas zonas para prevenir la asignación de direcciones de flujo allí donde no hay crestas claramente definidas, ya que eso podría causar problemas.

Este mapa llamado *mapa de bajo contraste* se computa en aquellas zonas donde los bloques con contraste suficientemente bajo son marcados. Este mapa separa el fondo de la imagen dela huella y localiza manchas y zonas con bajo nivel de tinta en la huella. Las minucias no son detectadas en estas zonas.

Una forma de diferenciar los bloques de bajo contraste de los que poseen crestas bien definidas es comparar la distribución de intensidad de sus píxeles. Por definición, habrá un rango dinámico muy pequeño en la intensidad de píxel de una zona de bajo contraste, ya que tanto la intensidad de las crestas como de los valles será muy débil. Sin embargo, en el caso de zonas con crestas bien definidas habrá píxeles cuya intensidad será muy débil en el centro de los valles, hasta píxeles casi negros en las crestas, formando así un rango de intensidad mucho más ancho que en las zonas de bajo contraste.

Ilustración 10: Resultados mapa bajo contraste [15]

En la figura 10 podemos apreciar los resultados generados por el mapa de bajo contraste, en el que las zonas con contraste suficientemente bajo como para imposibilitar la detección de minucias han sido marcadas con cruces blancas (esquinas superior-derecha e inferior-izquierda). En este caso, las zonas de bajo contraste se corresponden con el fondo de la imagen.

3.3.2.3 Mapa de bajo flujo

Es posible, durante la generación del mapa de dirección, que algunos de los bloques no posean un flujo de crestas dominante. Normalmente, estos bloques se encuentran en áreas de la imagen con baja calidad.

Inicialmente, a estos bloques no se les asigna ninguna orientación pero, posteriormente, estos bloques pueden recibir una orientación al interpolar el flujo de crestas de los bloques vecinos. El *mapa de bajo flujo* marca los bloques a los que, inicialmente, no se les puede asignar una flujo de crestas dominante.

Cuando las minucias son detectadas en estos bloques, se les asigna una calidad reducida, ya que han sido detectados como partes de la imagen con poca fiabilidad. En la siguiente imagen, las cruces blancas marcan bloques que no tienen un flujo de crestas dominante.

Ilustración 11: Resultados mapa bajo flujo [15]

3.3.2.4 Mapa de alta curvatura

Otras de las zonas problemáticas a la hora de detectar las minucias son aquéllas que presentan una alta curvatura. Estas zonas se corresponden normalmente con el *core (núcleo)* y las zonas de *deltas* de una huella. El mapa de alta curvatura marca bloques de la imagen que se encuentran en áreas de la huella con alta curvatura. Se usan dos formas de medida.

La primera, llamada *vorticity (vorticidad),* mide el cambio acumulativo en la dirección de flujo de crestas alrededor de todos los vecinos de un bloque.

La segunda, llamada *curvature (curvatura),* mide el máximo cambio de dirección en el flujo de crestas de un bloque y el de cada uno de sus vecinos.

Durante la detección de las minucias, a aquéllas detectadas en estos bloques se les asigna también una calidad reducida ya que estos bloques se marcan como partes de la imagen con poca fiabilidad. Las cruces blancas de la siguiente figura marcan zonas de la huella cuyas crestas presentan una alta curvatura, en este caso correspondiéndose con el *core.*

Ilustración 12: Resultados mapa alta curvatura [15]

3.3.2.5 Mapa de calidad

El último mapa que se produce durante este proceso es un mapa de calidad. Como se ha explicado, los mapas de bajo contraste, bajo flujo y alta curvatura apuntan a distintas regiones de baja calidad de la imagen. La información de estos mapas se integra en un mapa general que contiene 5 niveles de calidad como se muestra en la figura 13. La calidad que se asigna a cada bloque específico, se determina basándose en la proximidad de este bloque a aquéllos que están marcados en los mapas citados anteriormente.

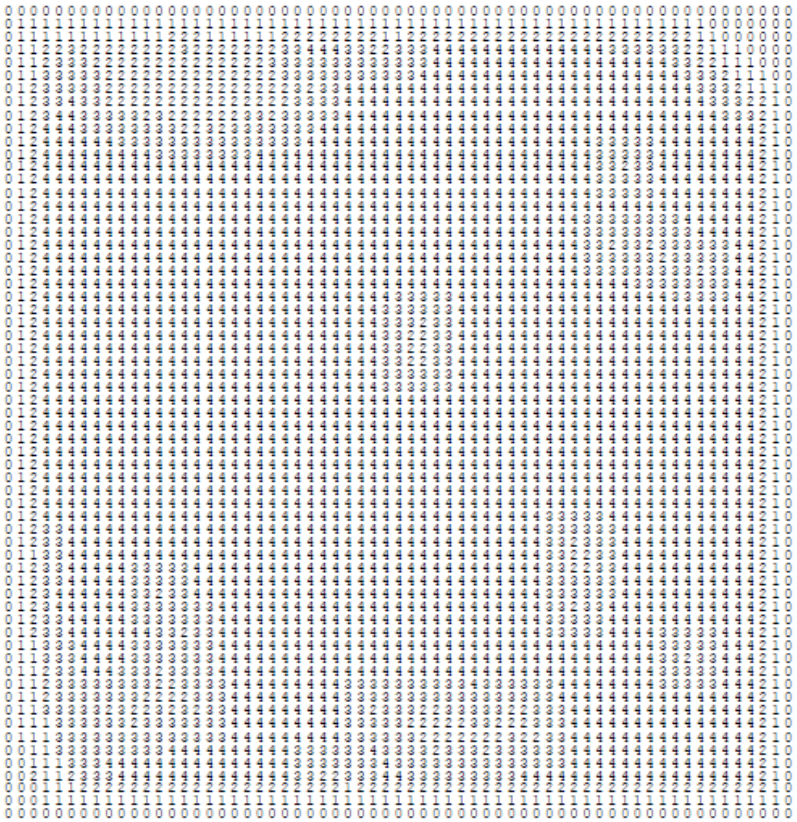

Ilustración 13: Resultados del mapa de calidad [15]

3.3.3 Binarización de la imagen

El algoritmo para detectar las minucias que se presenta está diseñado para trabajar con una imagen binaria (blanco y negro), donde los píxeles negros corresponden a las crestas y los blancos a los valles. Para poder crear esta imagen binaria, cada píxel de la imagen de entrada en escala de grises ha de ser analizado para determinar si se le debe asignar el color blanco o el negro. Este proceso se conoce como *binarización.*

A cada píxel se le asigna un valor binario basándose en la dirección del flujo de crestas asociada con el bloque en el que se encuentra. Si no se detecta ningún flujo de crestas, al píxel se le asigna el blanco. Si, por el contrario, se detecta flujo de crestas, las intensidades de los píxeles que rodean al píxel actual se analizan dentro de una matriz rotada como se indica en la siguiente figura:

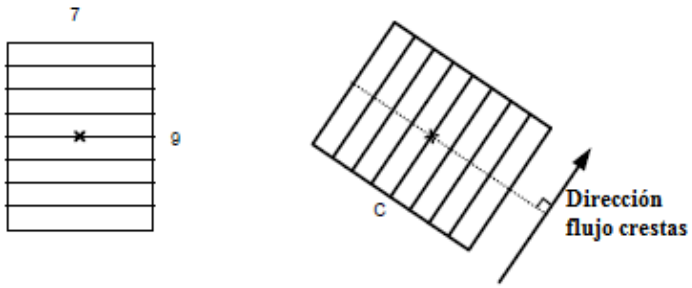

Ilustración 14: Matriz rotada usada para la binarización de la imagen [15]

Con el píxel de interés colocado en el centro de la matriz, como se puede ver en la imagen, la matriz se rota de forma que sus filas queden paralelas a la dirección de flujo de crestas en cuestión. Las intensidades de cada píxel en escala de grises son acumuladas a lo largo de cada fila rotada de la matriz, formando un vector de sumas de fila.

El valor binario que se asigna al píxel del centro se determina multiplicando el vector de sumas de la fila central por el número de filas de la matriz y comparando este valor con las intensidades en escala de grises acumuladas en toda la matriz. Si la multiplicación de la fila central es menor que la intensidad total de la matriz, entonces el píxel central se pone a negro, por el contrario, se pondría a blanco.

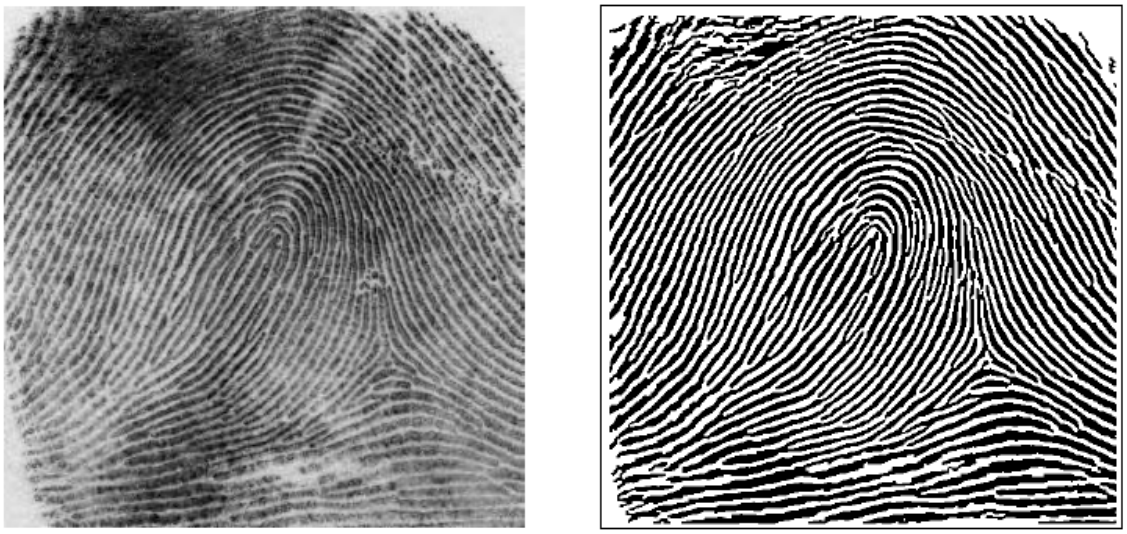

Ilustración 15: Resultados de la binarización [15]

En la figura superior podemos apreciar, a la izquierda, la imagen de huella original en escala de grises, y a la derecha la misma imagen binarizada.

La binarización es un paso crucial para el proceso de detección de las minucias. Se ha intentado crear la correcta comunión entre la necesidad de tener una imagen robusta y bien formada, y realizar un buen suavizado de las áreas de baja calidad sin dejar a la imagen exenta de calidad, de forma que sea apta para detectar las minucias.

3.3.4 Detección de las minucias

Este es el proceso en el que se detectan las minucias más fiables de la imagen. Consiste en realizar un escaneo de la imagen binarizada de la huella, identificando patrones de píxeles que indican el final o la división de una cresta. Como podemos comprobar en la siguiente ilustración, los patrones que se buscan son muy compactos:

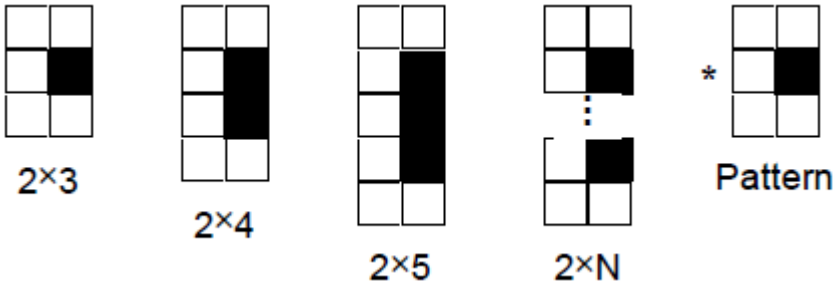

Ilustración 16: Patrones fin de crestas [15]

El patrón de la izquierda representa un conjunto de 6 píxeles en una configuración 2x3. El píxel negro de este patrón puede representar el final de una cresta que entrando al patrón desde la izquierda. Esta premisa puede aplicarse al resto de patrones de la ilustración, en los que podemos comprobar que el fin de cresta puede repetirse dentro de un conjunto de píxeles.

Siguiendo esta idea hay una serie patrones que se utilizan para detectar minucias de distintos tipos, estos patrones se ilustran a continuación:

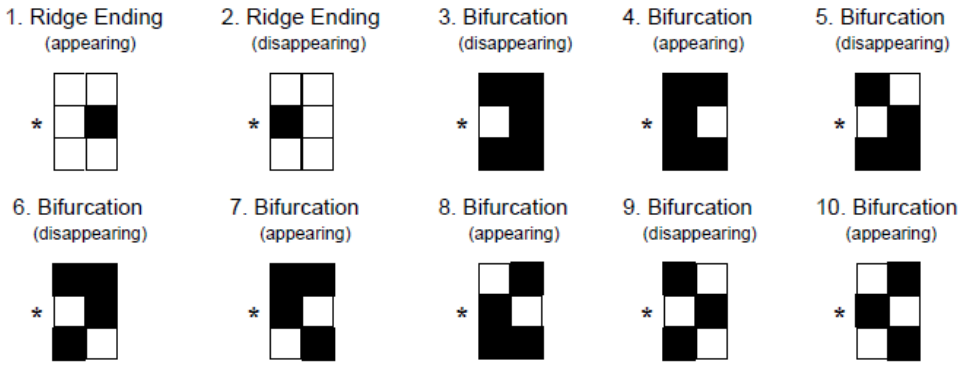

Ilustración 17: Patrones para la búsqueda de minucias [15]

3.3.5 Eliminación de falsas minucias

Como hemos visto en el apartado anterior, los patrones de minucias se pueden detectar con apenas 6 píxeles, esto facilita mucho que se pueda realizar un esquema de detección que minimice mucho la posibilidad de perder minucias reales. Sin embargo, muchas falsas minucias se incluyen en la lista de minucias candidatas. Este paso incluye la eliminación de *islas, lagos, agujeros, minucias en regiones de baja calidad de imagen, minucias a un lado, ganchos, solapamientos, minucias demasiado anchas y minucias demasiado estrechas (poros).* A continuación se hará una breve descripción de algunos de los principales pasos de eliminación.

3.3.5.1 Eliminación de islas y lagos

En este paso, bastas marcas de tinta (islas) y vacíos a lo largo de las crestas (lagos) son identificados y eliminados. Estas características de gran tamaño tendrán, típicamente, un par de minucias detectadas en lados opuestos tal como se muestra en la ilustración superior.

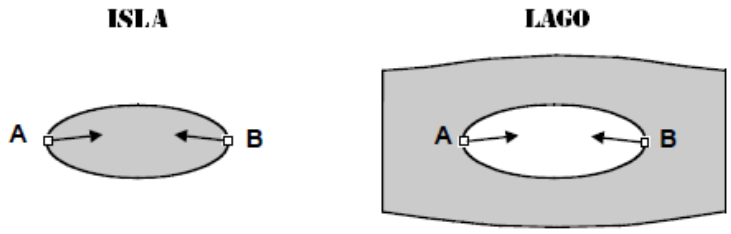

Ilustración 18: Isla y lago [15]

3.3.5.2 Eliminación de agujeros

Como podemos observar, un agujero se define prácticamente de la misma forma que una isla o un lago, solo que es más pequeño y tiene únicamente una minucia detectada a uno de sus lados.

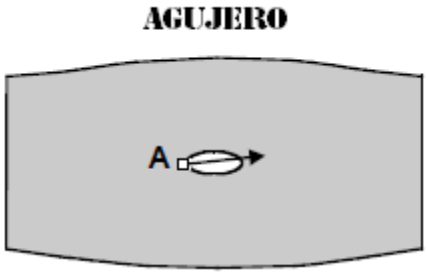

Ilustración 19: Agujero [15]

3.3.5.3 Eliminación o ajuste de minucias a un lado

Este paso busca cumplir dos objetivos. Primero, el ajuste de una minucia detectada de forma que se encuentre dispuesta más simétricamente en el final de una cresta o un valle. Durante el proceso, puede determinarse que no existe una forma simétrica clara donde se encuentra la minucia, lo que lleva al segundo objetivo, la eliminación de la minucia. Es el caso de la minucia B en la ilustración superior.

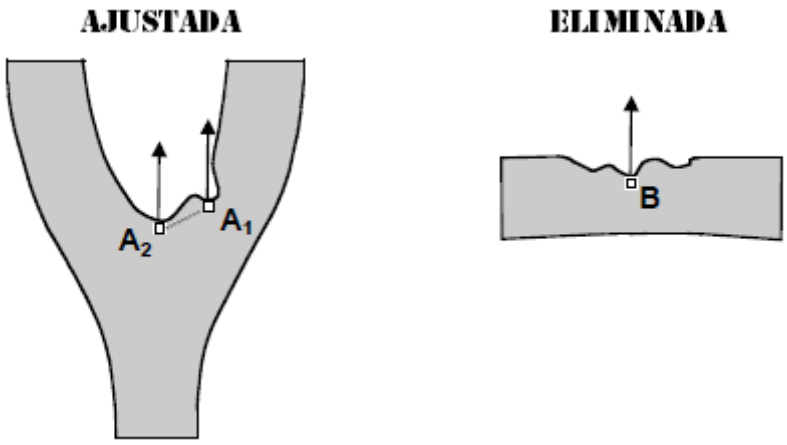

Ilustración 20: Minucias a un lado [15]

3.3.5.4 Eliminación de ganchos

Un gancho es una punta o espolón que sale hacia fuera por uno de los lados de una cresta o un valle. Esta característica típicamente tiene dos minucias de tipo opuesto, una en una pequeña porción de cresta y otra en un pequeño valle como se muestra en la figura.

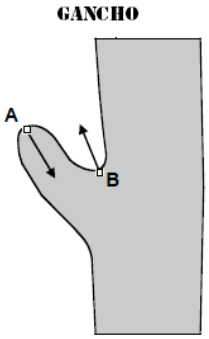

Ilustración 21: Gancho [15]

3.3.5.5 Eliminación de solapamientos

Un solapamiento es una discontinuidad en una cresta o un valle. Estas irregularidades normalmente se introducen en el proceso de impresión de la huella. Una rotura en una cresta provoca 2 falsos finales de cresta para ser detectados, mientras que la rotura de un valle causa 2 falsas bifurcaciones.

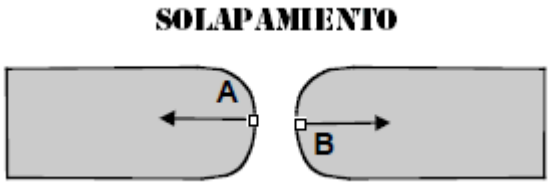

Ilustración 22: Solapamiento [15]

3.3.6 Recuento de crestas vecinas

A menudo, los algoritmos comparadores (*matchers*) usan información de las propias minucias detectadas. Esta información suele incluir el tipo de la minucia, su dirección y puede incluir también información relativa a sus minucias vecinas. En este paso se efectúa el recuento de las crestas que intervienen entre una minucia y sus vecinas más próximas.

3.3.7 Evaluación de la calidad de la minucia

Este paso tiene por objetivo asignar a cada minucia detectada una relación de calidad/fiabilidad ya que, a pesar de la cantidad de falsas minucias eliminadas, éstas pueden aun así persistir tras el proceso de eliminación.

Una robusta medida de calidad ayuda a asignar una menor calidad a las falsas minucias que resten en la lista de candidatas.

3.3.8 Archivo de salida de minucias

En el caso de la presente solución implementada, el algoritmo MINDTCT produce un archivo de texto final que recoge la información asociada a cada una de las minucias: posición X e Y, ángulo de dirección y calidad de la minucia. Este archivo de texto resultante se generará con una extensión ".xyt" y será utilizado por el algoritmo Bozorth3 para computar el resultado de comparación entre dos huellas.

3.4 Proceso detallado: BOZORTH3

A continuación se detallará el proceso seguido por el algoritmo Bozorth3 para realizar la comparación de dos huellas dactilares partiendo de sus archivos de salid ".xyt" hasta la computación de la puntuación de semejanza final.

3.4.1 Construcción de la tabla de comparación interna a la huella

El primer paso en el algoritmo Bozorth3 es computar medidas relativas entre cada minucia en una huella y el resto de minucias de la misma huella. Estas medidas relativas se guardan en una tabla de comparación de minucias y es lo que hace que el algoritmo sea invariante a rotación y traslación.

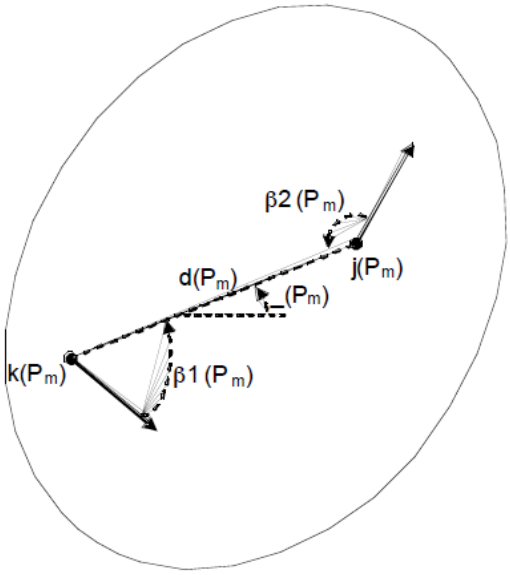

Ilustración 23: Comparación interna de minucias [15]

La figura 23 ilustra las medidas relativas que se efectúan. En este ejemplo existen dos minucias: la minucia K situada en la parte inferior izquierda y representada con un punto, y la minucia J situada en la parte superior derecha. La flecha que sale de cada minucia representa la orientación de las mismas.

Para tener en cuenta la posición relativa de traslación, la distancia d_{kj} se computa entre la posición de ambas minucias. Esta distancia permanecerá invariante independientemente de lo que se pueda girar o trasladar la imagen.

En cuanto a la orientación, el objetivo es medir el ángulo entre la línea que marca la orientación de las minucias y la línea que une ambas minucias. De esta manera, los ángulos permanecen constantes a dicha línea, independientemente de lo que se gire la imagen. Podemos ver estos ángulos en la ilustración superior: β_1 y β_2.

Para cada par de minucias comparadas, se introducirá una nueva entrada en la tabla de comparación que contendrá la información sobre la distancia entre minucias, los ángulos especificados anteriormente, y las propias minucias. Las entradas se ordenan en la tabla por distancia creciente hasta que se alcanza un límite superior prefijado, momento en el que la tabla se corta. Cada vez que se quieran comparar dos huellas, habrá que realizar una tabla de comparación para cada una de ellas.

3.4.2 Construcción de la tabla de compatibilidad interna a la huella

El siguiente paso es comparar las tablas de comparación de dos huellas separadas y buscara entradas compatibles entre ambas huellas. En la siguiente figura se muestran dos impresiones distintas de la misma huella con distinta rotación y distinta escala. En cada una de ellas se muestran dos minucias que se corresponden:

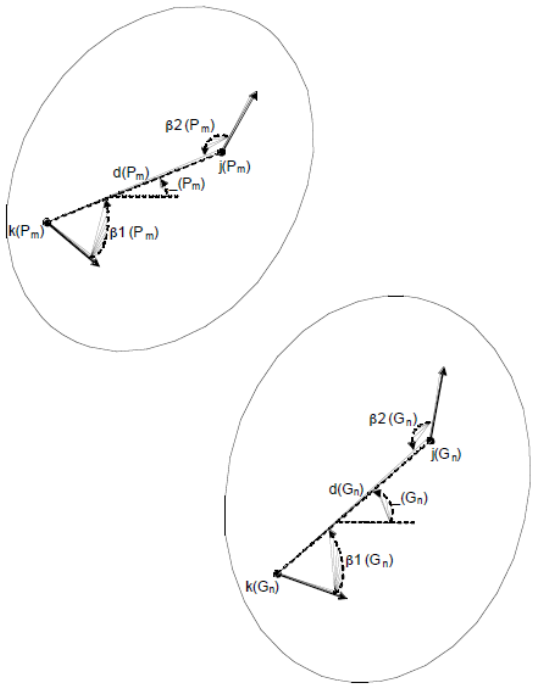

Ilustración 24: Medidas de un par de minucias compatible entre dos huellas [15]

La huella superior izquierda representa una imagen de entrada para la comparación. Todas sus minucias han sido comparadas por parejas y sus medidas relativas se han guardado en la tabla de comparación P. El subíndice "m" indica la entrada de la tabla en la que se encuentran guardadas sus medidas.

La huella inferior derecha ilustra una huella de la base de datos y presenta la misma notación, salvo que sus medidas relativas están guardadas en la tabla de comparación G en la entrada "n". A continuación se realizan 3 tests para comprobar si las dos entradas son compatibles. Primero un test para verificar que ambas distancias entre minucias se encuentren bajo una tolerancia especificada y los otros 2 tests para asegurar que los ángulos se encuentran bajo otra tolerancia determinada.

Si el resultado es positivo, entonces se efectúa una nueva entrada en la tabla de compatibilidad con la información relativa entre ambas minucias. Es por ello que en cada entrada de la tabla de compatibilidad se incluyen dos pares de minucias, indicando que la minucia K de la primera huella se corresponde con la minucia K de la segunda, y lo mismo para la minucia J.

3.4.3 Recorrido de la tabla de compatibilidad

En este punto del proceso, se ha construido una tabla de compatibilidad que consiste en una lista de asociaciones compatibles entre dos pares de minucias. Estas asociaciones representan enlaces en un gráfico de compatibilidad. Para determinar cómo de bien encajan entre ellos, un objetivo simple seria recorrer el gráfico en busca de la mayor trayectoria entre asociaciones compatibles. La puntuación final sería, entonces, el tamaño de dicha trayectoria, sin embargo, existen ciertas restricciones para conseguir este objetivo:

- La tabla de compatibilidad no es un gráfico coherente, sino una colección dispar de enlaces dentro de un gráfico.
- Cada nodo dentro del gráfico está potencialmente unido a otros nodos.
- No hay ningún "nodo raíz" claro que pueda llevar a alcanzar la máxima trayectoria.
- Oclusiones o vacíos dentro de cada una de las dos huellas que se comparan causaría discontinuidades en el gráfico.

Es en este punto cuando entra en juego el diseño del algoritmo realizado por Allan Bozorth. Éste se diseñó para que los recorridos de la tabla de compatibilidad se inicien desde distintos puntos de partida.

A medida que los recorridos avanzan, porciones del gráfico de compatibilidad son creadas mediante la unión de las entradas de la tabla. Una vez que los recorridos han concluido, las porciones compatibles se combinan y el número de entradas de la tabla enlazadas a través de esta combinación de porciones se suma para computar la puntuación de semejanza. Cuanto mayor sea el número de enlaces entre asociaciones compatibles, mayor será esta puntuación y mayor es la posibilidad de que las dos huellas pertenezcan a la misma persona.

4. Plataforma Android

Durante este apartado, hablaremos sobre la plataforma Android que es la que se ha escogido para desarrollar la aplicación realizada en este proyecto.

4.1 Conceptos generales

Android es un sistema operativo basado en Linux, cuyo uso principal está enfocado a dispositivos móviles como *smartphones, tablets…* El sistema es desarrollado por la Open Handset Alliance, liderada por Google.

En 2005, Google se hace con la firma Android Inc., primera desarrolladora de Android. Actualmente, las unidades vendidas de *smartphones* con Android como sistema operativo se encuentran en el primer puesto mundial, alcanzando cuotas de mercado de más del 50%.

La principal ventaja de Android, es que es código libre y posee una gran comunidad de desarrolladores trabajando en la constante mejora de las aplicaciones que utiliza este sistema y en el propio desarrollo del sistema operativo.

La principal fuente para adquirir aplicaciones para este sistema operativo es *Google Play*. Se trata de una tienda de aplicaciones en línea administrada por Google.

Por último, cabe destacar que Android utiliza el lenguaje de programación Java, que es sobre el cuál está basado el desarrollo del presente proyecto.

4.2 Versiones

A lo largo de los últimos años, el sistema operativo ha ido evolucionando rápidamente. Esto ha causado que se desarrollen numerosas versiones de Android en poco tiempo. Cada versión posee un nivel de API (Application Programming Interface) diferente. Una API es básicamente un protocolo utilizado a la hora de desarrollar software. La API puede incluir especificaciones para distintas rutinas, estructuras de datos, clases para diferentes objetos. A continuación se muestran las versiones de Android hasta la fecha, junto con la distribución en el mercado de cada una de ellas:

Plataforma	Nivel de API	%
4.x.x *Jelly Bean*	16-17	6,7%
4.0.x *Ice Cream Sandwich*	14-15	27,5%
3.x.x *Honeycomb*	12-13	1,6%
2.3.x *Gingerbread*	9-10	50,8%
2.2 *Froyo*	8	10,3%
2.1 *Eclair*	7	2,7%
1.6 *Donut*	4	0,3%
1.5 *Cupcake*	3	0,1%

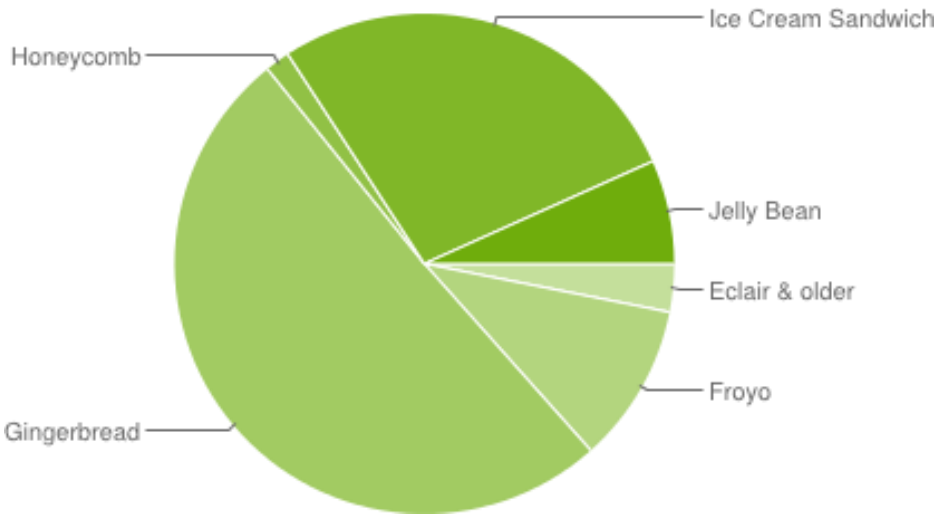

Ilustración 25: Versiones de Android y distribución en mercado actualmente [9]

Como podemos observar en la ilustración superior, a pesar de la aparición de nuevas versiones, la más utilizada sigue siendo la versión 2.3.X (Gingerbread). Es por ello que para el desarrollo de la aplicación para Android de este proyecto se ha escogido esa versión (nivel de API 10).

4.3 Ventajas

A continuación se analizarán distintas características por las cuales se ha elegido Android como plataforma para el desarrollo de la aplicación frente a otros sistemas operativos.

- *Implantación:* actualmente Android es dueño del mayor porcentaje del mercado de la telefonía. Por ello, realizar la aplicación para esta plataforma supone una mayor accesibilidad para los usuarios.

- *Código abierto:* Android es un sistema operativo totalmente libre y posee código abierto, lo que posibilita que cualquier desarrollado pueda, no solo crear sus propias aplicaciones, sino también mejorar las ya existentes dotando a sus dispositivos de un mayor rendimiento.

- *Libertad:* el sistema operativo brinda al usuario la posibilidad de instalar aquella aplicación que el usuario desee sin ningún tipo de traba, lo que permite tener un mayor control sobre su propio terminal.

- *Comunidad:* Android posee una gran comunidad de desarrolladores presentes por la Web. Por ello, la cantidad de ideas y consejos que se pueden encontrar a la hora de desarrollar código son ilimitados facilitando así a los usuarios el desarrollo de su propio código.

- *Multitarea:* Android tiene un sistema de multitarea que permite tener diversas aplicaciones en ejecución dejando en suspensión aquéllas que no se están utilizando y cerrándolas en caso de que no resulten ya útiles para evitar el consumo de memoria.

- *Personalización:* al tener un código abierto y libre, Android es totalmente personalizable permitiendo a los usuarios cambiar los fondos de pantalla, añadir *widgets…*, y a los fabricantes adaptar el sistema operativo para sus propios dispositivos.

4.4 Estructura

El sistema operativo Android es muy sencillo e intuitivo y posee una estructura basada en *activities.* Una *activity* es simplemente una cosa que el usuario puede hacer. Casi todas las *activities* interactúan con el usuario. A nivel de software, una *activity* no es más que una clase (en java). Debido a la interacción citada anteriormente la clase *activity* se encarga de crear una ventana (a nivel visual) la cuál se puede programar para que haga o muestre cualquier cosa.

Como se ha explicado, cada *activity* debe presentar al usuario una ventana. Para crear esta ventana es necesario asociar a la *activity* un *view (vista),* en la que se incluirán todas las formas, botones, colores, textos…que aparecerán en la *activity.* Se explicarán más adelante.

A continuación se adjunta un diagrama indicando el ciclo de vida de una activity en Android.

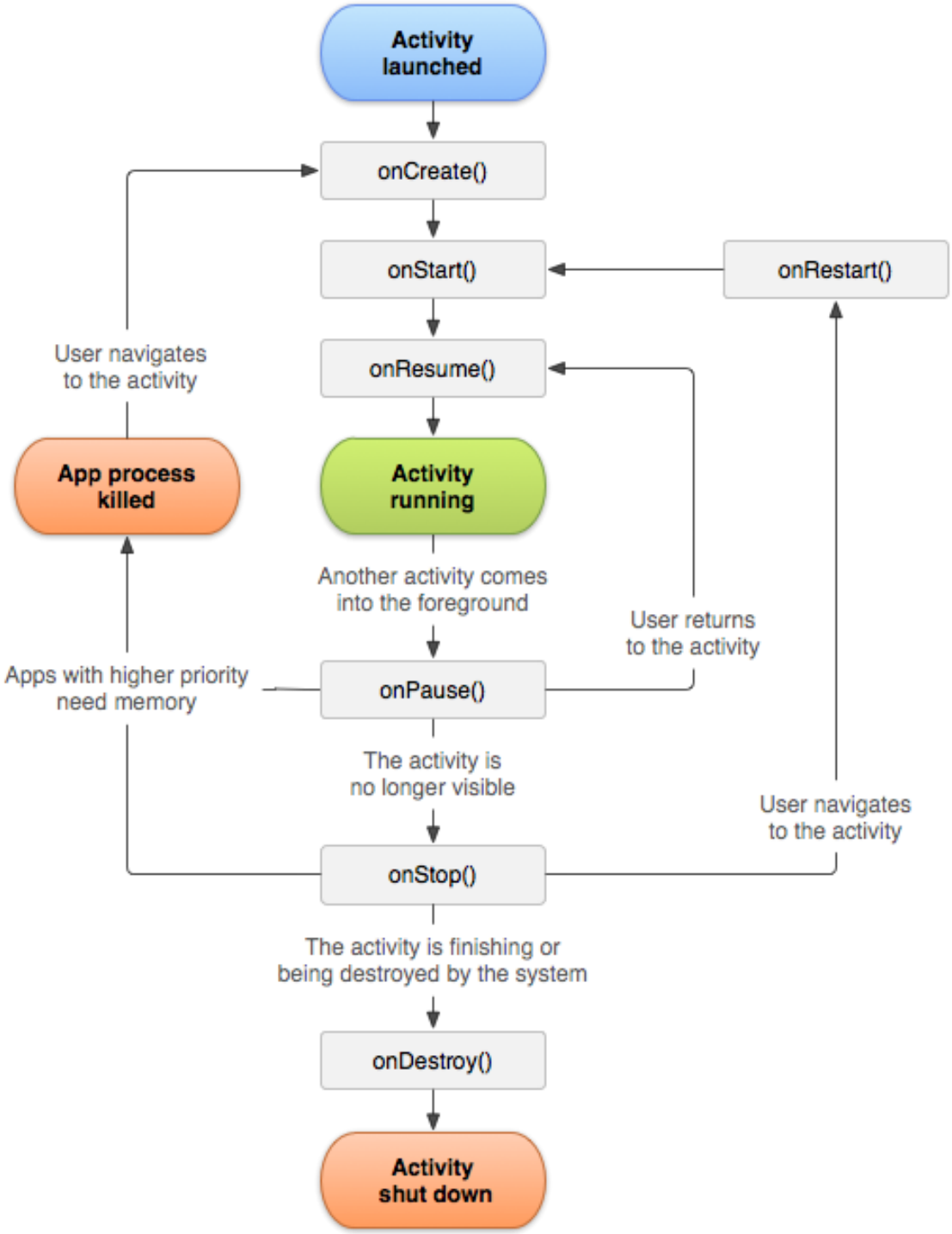

Ilustración 26: Ciclo de vida de una Activity [1]

En la figura 26 podemos observar el ciclo de vida anteriormente comentado, desde que la activity se lanza hasta que esta se cierrra, así como todos los caminos intermedios que la activity puede tomar.

5. Diseño de la aplicación: FprintApp

Durante este capítulo se explicarán los planteamientos seguidos para desarrollar la aplicación de la que trata el presente proyecto: *FprintApp.* Se hará una breve presentación de la estructura general de la aplicación para proceder a un análisis detallado en el siguiente capítulo de la memoria.

5.1 Descripción

FprintApp es una aplicación desarrollada para sistema operativo Android cuyo objetivo principal es el reconocimiento de huella dactilar para realizar verificación e identificación de candidatos.

La tecnología móvil actual no contempla la posibilidad de añadir un hardware de detección de huella dactilar a los dispositivos móviles Android como *smartphones* y *tablets.* Es por ello que la aplicación no puede trabajar con huellas latentes tomadas por el propio teléfono, de forma que se trabaja con imágenes de huellas previamente capturadas por dispositivos externos.

A lo largo del desarrollo de la aplicación, siempre se ha intentado que ésta posea colores vistosos y que sea intuitiva para el usuario. En todas las pantallas de la aplicación se incluye un botón de ayuda que guiará al usuario en todo momento explicando los pasos que ha de seguir para completar la acción deseada en cada momento.

Por otro lado, todos los botones que presenta la aplicación han sido escogidos de forma que el usuario pueda entender cuál es la función que desempeñan simplemente con una vista rápida.

En la siguiente ilustración se incluyen dos capturas de pantalla tomadas desde un dispositivo móvil Android para demostrar la sencillez de la aplicación y sus formas vistosas para la máxima comodidad de los usuarios.

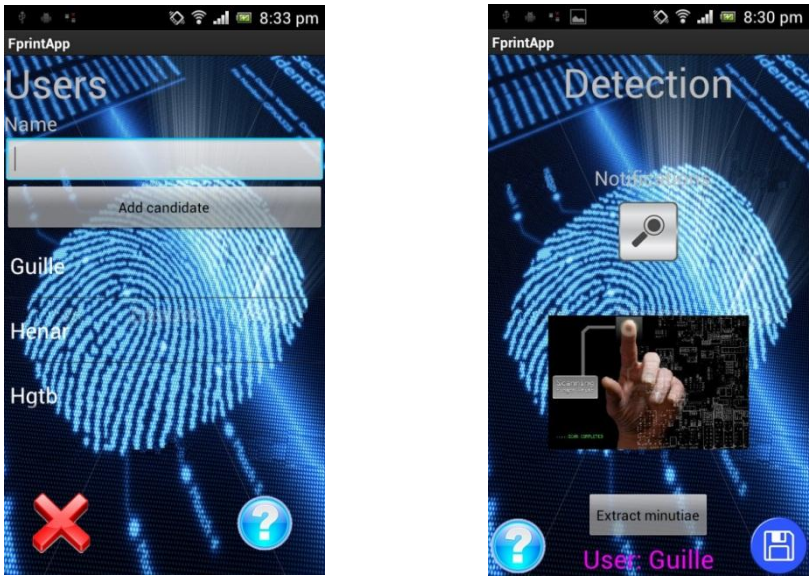

Ilustración 27: Capturas de pantalla de la aplicación FprintApp

Se puede observar en las anteriores capturas que se ha optado por un diseño con colores vistosos e imágenes intuitivas. Esto es así para eliminar cualquier posible signo de complejidad a la hora del manejo de la aplicación. De esta forma, la FprintApp conseguirá una mayor aceptación entre todos los tipos de usuarios.

5.2 Diagrama de flujo

A continuación se incluirá un diagrama de flujo con el que se podrán ver claramente las trayectorias que puede seguir la aplicación dependiendo de la *activity* a la que se esté accediendo en cada momento.

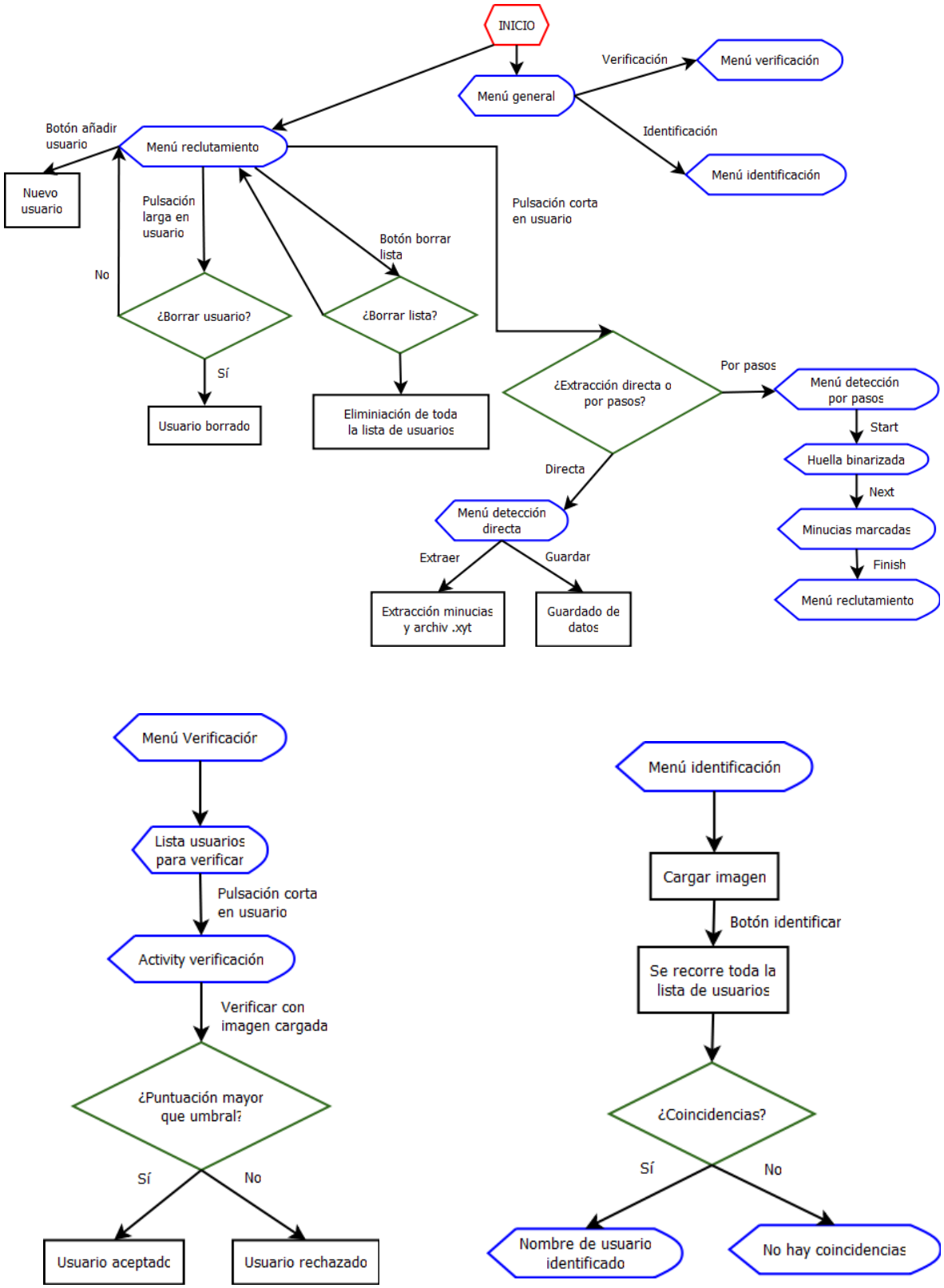

Ilustración 28: Diagrama de flujo de la aplicación

En la ilustración superior podemos observar el diagrama de flujo de que nos muestra la funcionalidad de la aplicación de forma muy general. Para mayor claridad, se han excluido la representación de algunos views básicos como el botón *Exit*, que finaliza la aplicación de forma global. En los siguientes apartados se detallará con más profundidad toda la funcionalidad de FprintApp.

5.3 Funcionalidades

En este apartado se expondrá la funcionalidad de cada *activity* de la aplicación de forma detallada. Durante este apartado de la memoria, se hará usos de términos específicos del lenguaje de programación Java basado en Android. Para mayor comodidad, se hará una breve descripción de cada término en el glosario de términos incluido en las primeras páginas de la memoria.

5.3.1 Views comunes a todas las activities

Primeramente y para no repetir la misma explicación a lo largo de las distintas activities, explicaremos aquí aquellos views que son comunes a la mayoría de las activities de la aplicación y que realizan una función idéntica en cada una de ellas.

- *Botón de ayuda (interrogación):* la función de este botón es únicamente invocar a una activity basada en cuadro de texto en el que se incluyen todas las indicaciones necesarias para que el usuario pueda desempeñar todas las funciones que ofrece la activity en cuestión, en este caso, la del menú de reclutamiento. Este botón se incluirá en el resto de activities de la aplicación y presentará una funcionalidad idéntica en cada una de ellas, por lo que no se volverá a definir su función en la explicación de las activities restantes.

Ilustración 29: Botón ayuda

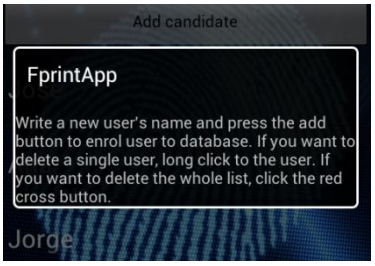

Ilustración 30: Texto ayuda

- *Botón cargar imagen (lupa):* en las activities de detección, verificación e identificación necesitaremos primeramente cargar la imagen de una huella dactilar para poder procesarla y realizar la acción que se requiera dependiendo de la activity seleccionada. Por ello se suministra al usuario un intuitivo botón que le permitirá seleccionar una imagen de su galería personal.

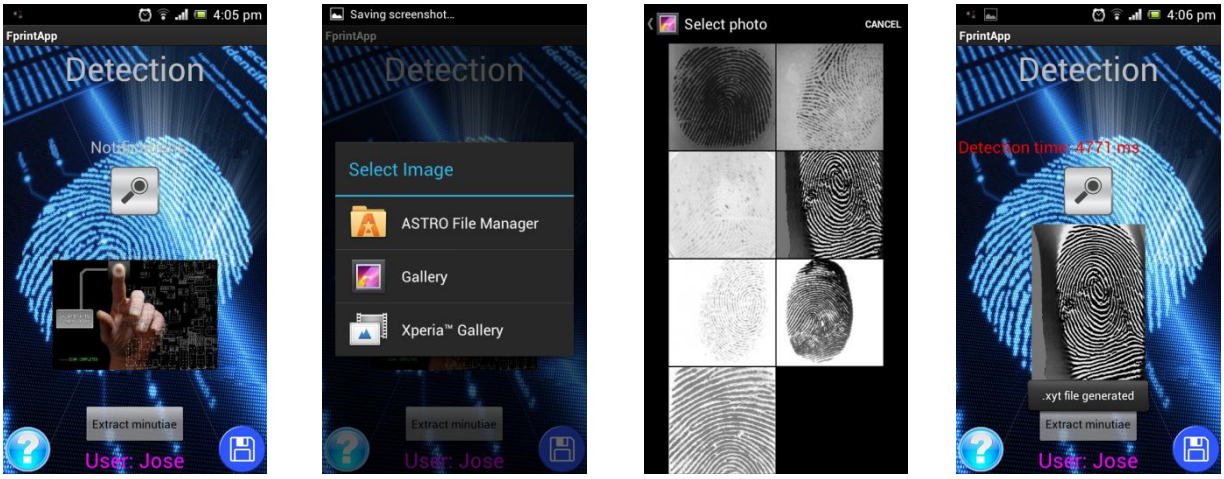

Ilustración 31: Proceso selección imagen.

Esta sucesión de imágenes muestra el proceso que se sigue para cargar una imagen de huella. Se comienza pulsando el botón de cargar imagen (lupa). Después una ventana emergente permitirá elegir el método de selección, en este caso se ha seleccionado la galería. Una vez dentro de la galería se escoge una imagen de huella dactilar y, finalmente, ésta aparecerá representada en la activity.

- *Botón de guardado:* este botón es únicamente común a las activities de detección directa y detección por pasos. En estas activities se procede a obtener un archivo de minucias a partir de una imagen de huella para un usuario concreto. El botón de guardado permitirá que, una vez extraído, el archivo de minucias sea guardado en un fichero personalizado para el usuario concreto para el que se esté detectando la minucia. De esta forma, el archivo de minucias quedará permanentemente asignado a ese usuario, eliminando la necesidad de tener que volver a extraer un nuevo archivo cada vez que se quiera realizar una acción con ese usuario.

Ilustración 32: Botón guardado.

En la imagen de la derecha podemos observar el resultado de pulsar el botón de guardado. Una vez pulsado, la activity de detección, ya sea por pasos o directa, se cerrará automáticamente devolviendo la aplicación al menú de reclutamiento. Un *toast (mensaje emergente en Android)* nos indicará si el archivo se ha guardado correctamente como y el nombre del usuario al que se ha añadido. Podemos apreciar el resultado de pulsar el botón de guardado en la imagen de la derecha.

Ilustración 33: Resultado botón guardado.

- *Tiempos de proceso:* en las activities de verificación, detección directa y detección por pasos se realiza una medida del tiempo de proceso. En detección por pasos, esta medida se realiza tanto para el proceso de detección general como para cada uno de los subprocesos que se incluyen en la activity.

Ilustración 34: Timer detección directa.

5.3.2 Menú principal

La primera activity que encontramos al iniciar la aplicación es un menú en el que se podrá seleccionar la función a realizar por la aplicación, ya sea identificación, verificación o reclutamiento. En la siguiente ilustración podremos visualizar claramente la interfaz gráfica establecida para este menú.

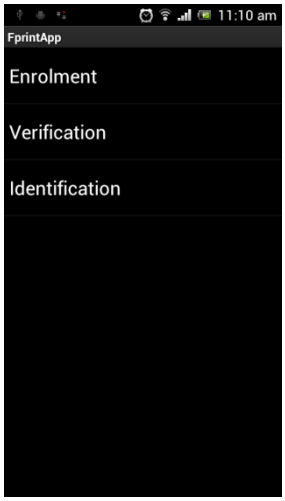

Ilustración 35: Captura pantalla menú principal.

Como se puede observar, el menú principal se compone de la implementación de un *ListView*, que es básicamente una lista de elementos que se pueden desplazar hacia arriba y hacia abajo.

Para acceder a cada una de las funciones que permite esta aplicación, lo único que habrá que hacer será pulsar sobre uno de los tres elementos implementados y, automáticamente, se accederá a la activity en cuestión.

5.3.3 Menú reclutamiento (Enrolment)

Tras seleccionar el elemento *Enrolment (reclutamiento)* se accederá al menú de reclutamiento, en el que se podrán añadir o eliminar usuarios en la base de datos y adjudicarles un archivo de minucias *.xyt* personalizado para cada uno de ellos.

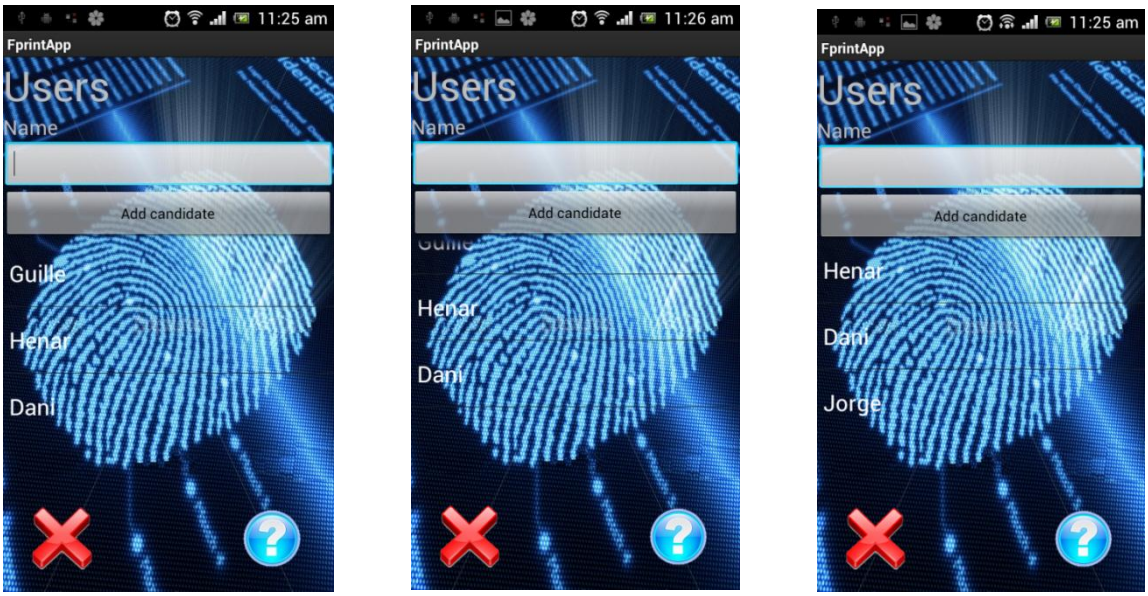

Ilustración 36: Menú reclutamiento desplazando elementos

En las figuras superiores se pueden observar todos los views que contiene el menú de reclutamiento. También se puede apreciar que, para la lista de usuarios, se ha optado de nuevo por la implementación de un ListView y, en la sucesión de las tres imágenes se observa cómo la lista es desplazada hacia arriba para mostrar la totalidad de los usuarios incluidos dentro de la lista, pudiendo acceder así al usuario "Jorge" que, como se ve en la primera imagen, quedaba fuera de la pantalla.

A continuación, se hará una lista con todos los views contenidos en esta activity para proceder a una explicación profunda de cada uno de ellos:

Ilustración 37: Menú reclutamiento, campo de texto.

- *Campo de texto (TextField):* se trata de uno de los views más básicos que podemos encontrar en Android. No es más que un espacio en el que podemos escribir cualquier texto a través del teclado táctil del dispositivo Android en cuestión para, posteriormente, procesarlo de la manera que se crea conveniente. En la figura podemos observar el resultado de pulsar sobre el campo de texto. El teclado emerge automáticamente y el resto de views se ven

"arrastrados" por él. Sin embargo esto no es un problema ya que en el momento en el que se pulse el botón *Add candidate* el usuario quedará registrado en la lista, el teclado desaparecerá y el resto de views volverán a su posición original. De esta forma se consigue un entorno sencillo en el que el usuario no ha de preocuparse de ningún aspecto técnico de la activity.

- *Borrar lista (aspa roja):* este view se conoce como *botón (button)* y, como vemos, su aspecto gráfico es totalmente personalizable. En este caso se ha procedido a adjudicarle una imagen de un aspa roja que resulta muy intuitiva para cualquier usuario. Aun así, se ha incluido un botón de ayuda que se explicará más adelante. Tras pulsar este botón, aparecerá un *AlertDialog*, view que consiste en un cuadro de diálogo que presenta dos opciones para realizar y el usuario tendrá que seleccionar una de ellas. En este caso nos dará las opciones de borrar o no la lista de usuarios al completo como podemos apreciar en la imagen. Si elegimos no borrarla, se volverá de nuevo al menú de reclutamiento original, y si la borramos, el menú de reclutamiento aparecerá con una lista de usuarios vacía.

Ilustración 38: Botón borrar lista.

Ilustración 40: AlertDialog borrar lista.

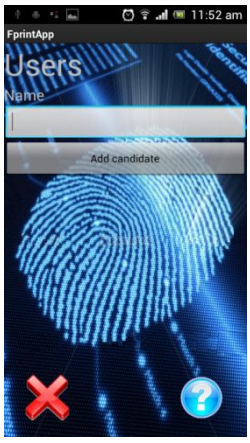

Ilustración 39: Resultado borrar lista.

- *Asignación de huella (pulsación corta sobre usuario):* para acceder a la activity de detección de minucias, lo que habrá que hacer es realizar una pulsación corta sobre uno de los elementos de la lista de usuarios. Tras realizar esta acción, aparecerá un nuevo AlertDialog que dará la opción de realizar una detección de minucias directa o paso a paso. Después de seleccionar una de las dos opciones se abrirá la activity de

Ilustración 41: AlertDialog extracción minucias.

detección de minucias en la que podremos asignar el archivo de minucias extraído al usuario que se haya seleccionado.

- *Eliminar usuario (pulsación larga sobre usuario):* si se realiza una pulsación prolongada sobre uno de los elementos de la lista de usuarios, un nuevo AlertDialog presentará las opciones de borrar o no borrar el sujeto seleccionado. Si se elige borrarlo, el usuario desaparecerá de la lista y ésta autoajustará las posiciones para evitar que queden espacios en blanco entre los elementos.

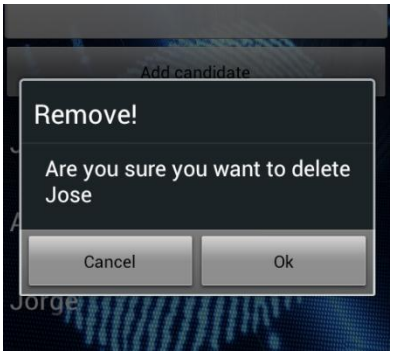

Ilustración 42: AlertDialog detección de minucias.

5.3.4 Activity detección directa de minucias

Una vez realizada una pulsación corta sobre uno de los usuarios de la lista del menú de reclutamiento, un AlertDialog emergerá y otorgará la posibilidad de efectuar una detección de minucias para el usuario de forma directa o paso a paso. En este apartado se explicará la activity en la que se realiza la detección directa. En la imagen de la derecha se puede apreciar el aspecto del AlertDialog que nos introducirá en la activity de detección directa o por pasos dependiendo de la elección tomada. A continuación se muestra una vista general de la activity de detección directa y, posteriormente se hará una enumeración de los views que contiene.

Ilustración 43: AlertDialog detección de minucias.

Ilustración 44: AlertDialog detección de minucias.

- *Botón extraer minucia (Extract minutiae):* como se puede observar en la imagen de la vista general de la activity, éste es el botón central de la aplicación. Una vez cargada una imagen de huella dactilar podremos pulsar este botón y, automáticamente, se ejecutarán todos los pasos del algoritmo MINDTCT para generar el archivo ".xyt" que contendrá la información acerca de todas las minucias detectadas. Una vez finalizado todo el proceso, un *toast* nos indicará que las minucias han sido extraídas correctamente.

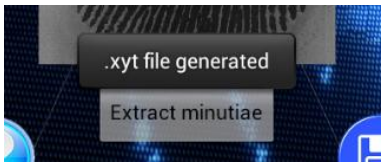

Ilustración 45: Toast resultado detección directa.

- *Botón de guardado:* ya explicado anteriormente.

- *Tiempo de proceso:* en el caso de esta activity, la cuenta de tiempo generada al final del proceso se corresponde con el tiempo usado para procesar la huella desde la carga hasta la generación del archivo ".xyt".

Ilustración 46: Tiempo proceso detección directa.

- *Indicador usuario (User):* un textview indicará cada vez que se entre en la activity el nombre del usuario para el que se están extrayendo las minucias.

Ilustración 47: Usuario actual.

5.3.5 Activity detección por pasos

Para acceder a esta activity, se deberá elegir la opción "StepByStep" en el AlertDialog emergente en el momento de pulsar un usuario en el menú de reclutamiento (figura 43). Dada la notable semejanza entre esta activity y la de detección directa se enumerarán aquellos views que presentan una funcionalidad idéntica en ambas y se explicarán los nuevos, los cuáles son las distintas vistas de cada paso incluido por separado.

Ilustración 48: Vista con huella activity detección por pasos.

- *Botón de guardado.*
- *Indicador usuario (User).*
- *Botón inicio/siguiente/finalizar (Start/Next/Finish):* este es el botón que permite inicializar los dos pasos cruciales que se han representado en esta activity y avanzar entre ellos ya que, como se va a ver en las siguientes capturas de pantalla, una vez pulsado Start, el botón pasará a tener el nombre de Next y, finalmente, el de Finish para acceder al último paso. A continuación se explicará cada uno de ellos detenidamente.
- *Paso 1 (binarización):* cuando pulsemos el botón Start el imageview que contiene la imagen de la huella pasará a mostrar la imagen de la huella binarizada. Asimismo se adjunta una breve descripción encima de la imagen acerca del proceso de binarización. En la parte inferior de la activity se incluye la medida del tiempo que ha llevado la binarización del imagen, previamente en escala de grises.

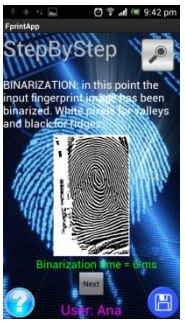

Ilustración 49: Paso 1, binarización de la imagen.

- *Paso 2 (señalización de las minucias):* en este paso se señalan todas las minucias detectadas sobre la imagen, excluyendo aquéllas descartadas en los procesos de eliminación. También se incluye una pequeña descripción y la cuenta del tiempo empleado en la detección y descarte de minucias. Una vez realizado este proceso, un toast nos indicará que el archivo ".xyt" ha sido generado al igual que en la detección directa.

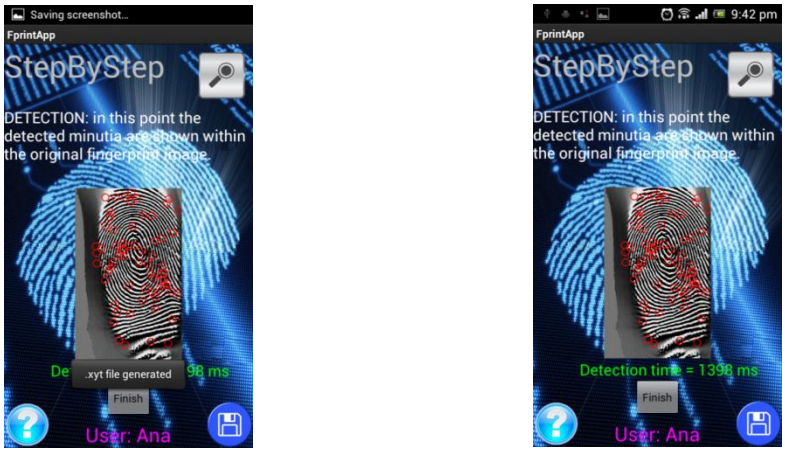

Ilustración 51: Paso 2, señalización de las minucias.

- *Paso final (recuento general):* una vez pulsado el botón Finish, se accederá al paso final, en el que se presenta una cuenta de tiempo de todo el proceso desde la carga de la imagen hasta la generación del archivo ".xyt". Una vez que se llega a este punto, el botón Finish se transformará en el botón "Exit without save", el cual permitirá abandonar la activity sin guardar el archivo para ese usuario. Aun así seguirá existiendo la posibilidad del botón de guardado.

Ilustración 52: Paso final, recuento general.

5.3.6 Activity verificación

En esta activity se puede realizar el proceso de verificación. Primeramente se accederá a la lista de usuarios para seleccionar aquél con el que se quiera realizar la verificación. Una vez seleccionado se entrará a la activity de verificación donde, tras cargar una huella, el usuario será evaluado como válido o inválido dependiendo si el algoritmo BOZORTH3 genera un resultado superior o inferior respectivamente al umbral establecido.

Ilustración 53: Vista general activity verificación.

- *Botón verificar usuario (Verify user):* se trata del botón que, una vez cargada una imagen, se generará automáticamente el archivo ".xyt" de esta y se aplicará el algoritmo BOZORTH3 entre este archivo y el archivo previamente guardado mediante la función de "Enrolment" del usuario en cuestión. Como resultado se obtendrá "Access granted" o "Access denied" dependiendo si la imagen corresponde o no al usuario en cuestión. Además se mostrará el tiempo empleado en la verificación.

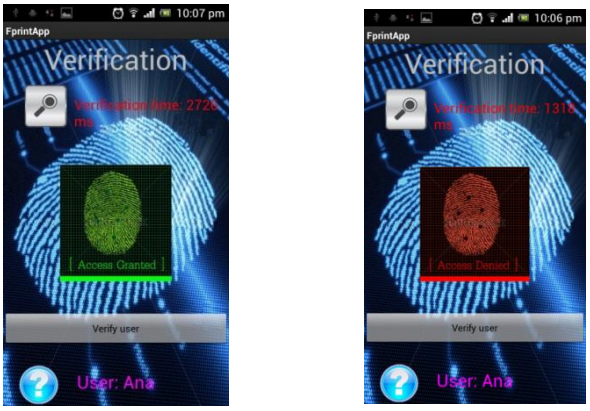

Ilustración 54: Resultados verificación.

5.3.7 Activity identificación

En esta activity se podrá cargar una imagen y comprobar a qué usuario de la lista corresponde mediante la aplicación del algoritmo BOZORTH3 a cada uno de los usuarios de la lista de forma automática. El resultado será el nombre del usuario si se ha detectado, o el mensaje de "identidad nula" si no existe un usuario con esa huella dactilar en la base de datos.

Ilustración 55: Vista general activity identificación.

- *Botón identificar (Identify user):* una vez cargada una imagen y, tras pulsar este botón, se podrán producir dos resultados diferentes. Que se haya encontrado al usuario al que pertenece esa huella o que no haya ningún usuario con dicha huella en la base de datos. A continuación se muestra un ejemplo de cada posible resultado.

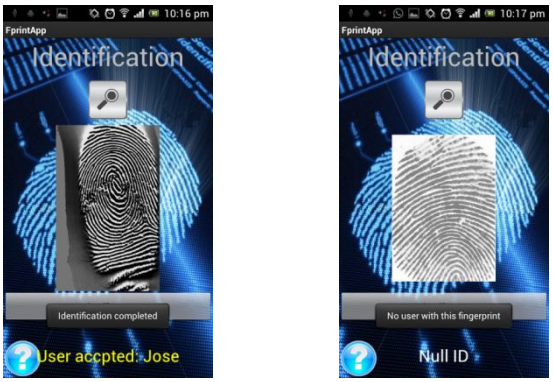

Ilustración 56: Distintos resultados identificación.

Como se puede observar, independientemente del resultado de la verificación, un toast informará con una pequeña descripción de lo ocurrido tras la aplicación del algoritmo BOZORTH3.

- *Botón nuevo/salir (New/Exit):* tras realizar la identificación, si se vuelve a pulsar el botón de identificar (Identify user) un AlertDialog proporcionará las opciones de salir de la activity o resetearla para proceder a una nueva identificación.

Ilustración 57: AlertDialog identificación.

6. Desarrollo de la aplicación

En este apartado se hablará de la parte más técnica del proyecto. Se introducirá el entorno de desarrollo utilizado para la elaboración del proyecto y se explicarán a nivel de código los detalles implementados más destacables tanto a la hora de traducir el proyecto de reconocimiento de huella dactilar de C# a Java como en la creación de la aplicación Android que sustenta este proyecto.

6.1 Entorno de desarrollo: Eclipse

Eclipse es un entorno de desarrollo integrado multiplataforma. Con este software es posible programar y realizar aplicaciones en diversos lenguajes.

Eclipse fue desarrollado por IBM, sin embargo, actualmente se encuentra desarrollado por la organización independiente *Fundación Eclipse*, la cual no tiene ánimo de lucro y fomenta una comunidad de código abierto.

A continuación se incluye una vista general de este programa con un proyecto abierto. Como se puede observar, el entorno está organizado en cómodas ventanas que se pueden modificar al antojo del usuario. Pudiendo quitar las ventanas, añadir otras o moverlas. A la izquierda del todo se puede observar el explorador de paquetes. Es en esta ventana donde podemos seleccionar cualquiera de los proyectos que tengamos abiertos y navegar por todos sus archivos. Si seleccionamos un archivo *.java* o *.xml* de un proyecto, se abrirá en la ventana principal, que es en la que se realiza el código. En la parte derecha hay dos ventanas, la superior es una lista de tareas totalmente configurable y la inferior es una ventana que muestra las funciones y variables que poseen todos los archivos del paquete que se esté utilizando actualmente.

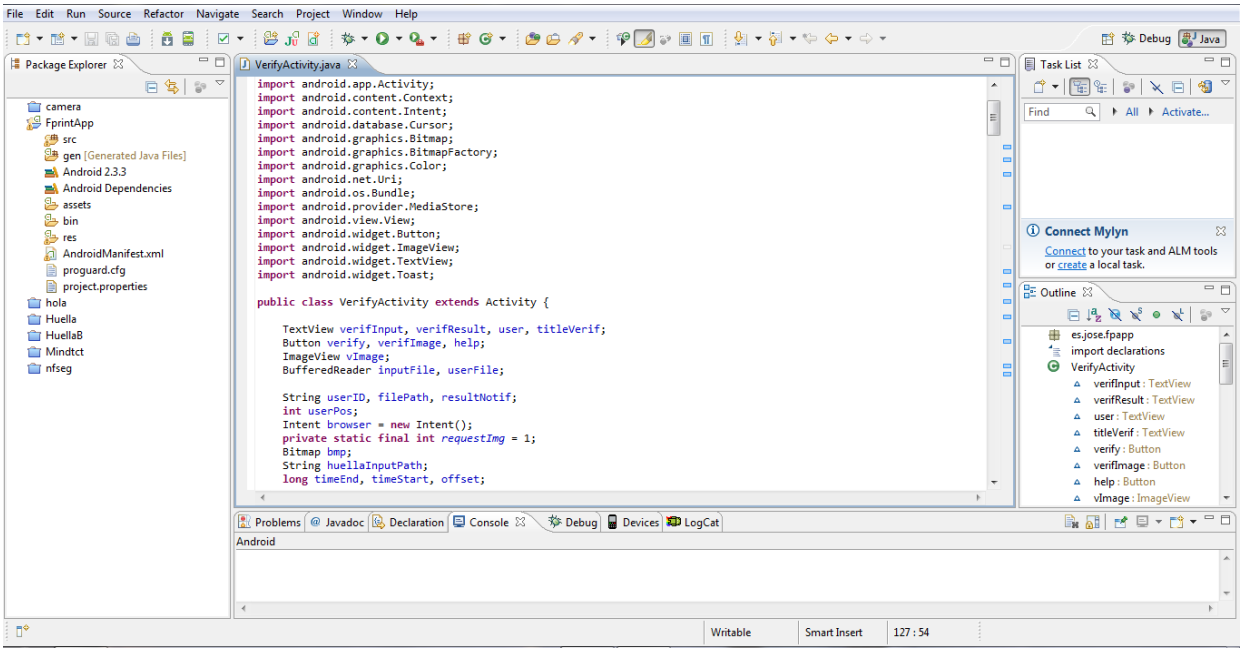

Ilustración 59: Vista general IDE Eclipse.

Sin embargo, falta un paso más para poder trabajar con Android dentro de Eclipse, y es añadir el SDK de Android. Un SDK (software development kit) es un kit de desarrollo de software, es decir, un conjunto de herramientas de desarrollo software que permiten al programador realizar aplicaciones para un sistema concreto. El SDK Android incluye ciertas herramientas como bibliotecas especializadas para uso en Android, así como códigos de ejemplo y muchas otras utilidades.

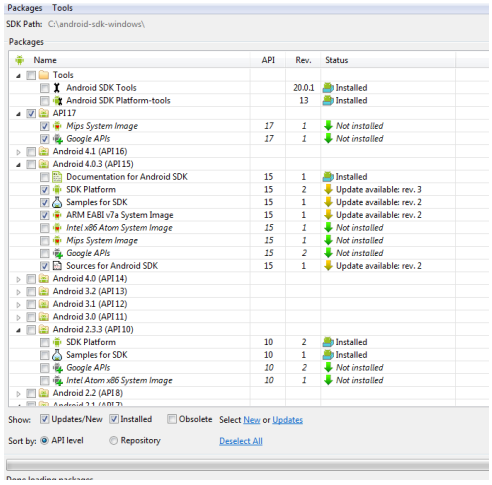

Ilustración 61: Android SDK.

En la figura se puede observar el menú del SDK de Android dentro de Eclipse. En el aparecen todas las herramientas y utilidades disponibles para cada una de las APIs de Android. El usuario puede descargar aquéllas que más necesite pasando a poder disfrutar de ellas en el momento en que la descarga finalice ya que se integrarán automáticamente en el software Eclipse.

Además de todo lo anterior, es necesario integrar el lenguaje Android dentro de Eclipse, ya que por defecto no viene con este lenguaje de programación instalado. Esto se puede realizar fácilmente consultando la página de Android Developer, donde se facilita todo lo necesario para configurar Android en Eclipse.

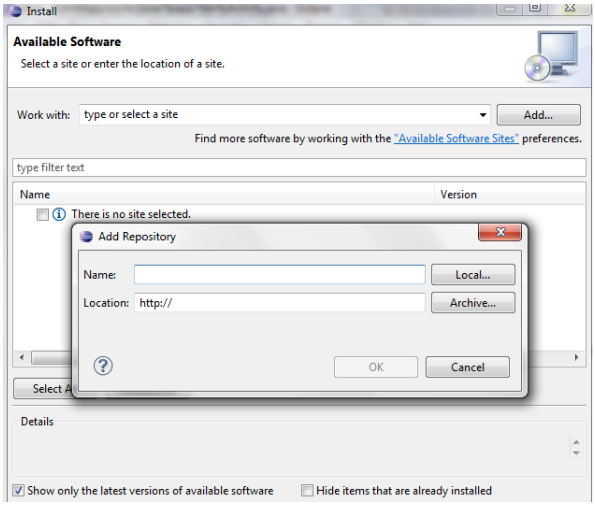

Ilustración 62: Eclipse Software.

En la figura superior se puede ver la ventana a través de la cual podemos añadir repositorios a Eclipse. En este caso se utilizó el nombre y la localización proporcionada por la página Android Developer para poder descargar el software necesario para instalar Android enn Eclipse.

En la imagen de la derecha se pueden apreciar las partes que componen un proyecto de Android una vez creado a través de Eclipse. A continuación se citarán las partes más destacables del proyecto:

- *Src:* se trata de la carpeta que incluye todas las clases (*.java*) que contendrá el proyecto. En estas clases es en la que se programa el código de toda la aplicación.

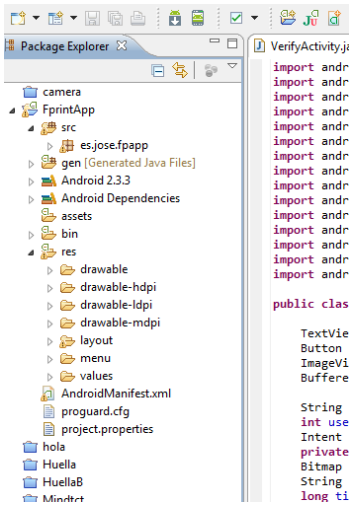

Ilustración 63: Proyecto Android.

- *Res:* es la carpeta *resources*, en ella se encuentra todo el apartado gráfico de la aplicación. Por ejemplo, en las carpetas *drawable* se incluyen todos los archivos de imagen usados para la confección de botones, fondos de pantalla, etc…. La carpeta *menú* contiene la distribución de los elementos del menú inflable que se puede sacar en la pantalla del menú principal. La carpeta *values* contiene los valores de la mayoría de los *strings (cadenas de texto)* usadas a lo largo de la aplicación. En la carpeta *layout* se encuentran todos los archivos *.xml.* Estos archivos contienen la distribución gráfica de cada una de las activities y se puede programar mediante código o a través de una herramienta que nos permite ver gráficamente el diseño de la activity como muestra la imagen inferior.

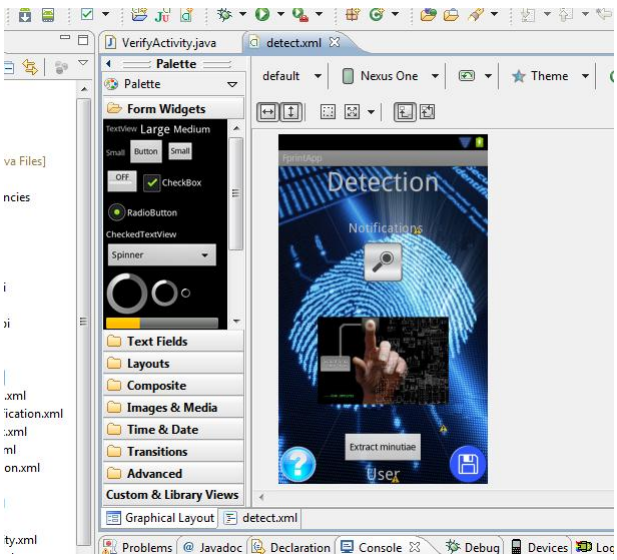

Ilustración 64: Layout de una activiy.

- *AndroidManifest.xml:* se trata de un archivo *.xml* de texto en el cual se han de declarar todas las activities que posea el proyecto en cuestión para que la aplicación pueda reconocerlas y acceder a ellas. Además se especifica tanto el nombre que recibirá el *Intent* que las llame, como otros parámetros de interés como establecer la prioridad a la hora de instalar, si se quiere hacer en almacenamiento interno o externo u orientación de las activities. Un *Intent* no es más que una clase predefinida de Android usada para llamar a alguna activity dentro de otra

Ilustración 65: Android Manifest.

6.2 Semejanzas y diferencias con C#

En este proyecto, la implementación de los algoritmos BOZORTH3 y MINDTCT se ha elaborado a partir de la idea original de otro proyecto realizado con anterioridad en lenguaje C#. La semejanza encontrada a la hora del uso de funciones de biblioteca es bastante alta entre ambos lenguajes, difiriendo éstos en la mayoría de los casos en el uso de letras mayúsculas o minúsculas, como por ejemplo las funciones de la biblioteca *Math* en Java utilizadas para diversos cálculos matemáticos. Poniendo como ejemplo la función usada para hallar el absoluto entre dos números, se encuentra que en C# se declara como *Abs()*, mientras que en Java es *abs()*. Estas sutiles diferencias aparecen constantemente durante la traducción de código. Sin embargo, existen algunas diferencias más notables, las cuáles se citan a continuación.

- *Paso por referencia:* esta es una de las diferencias más tediosas que se han encontrado. A lo largo de la implementación de los algoritmos es necesario modificar algunos atributos de funciones y variables en su dirección de memoria. Para entender mejor esta explicación se adjunta a continuación un ejemplo de la declaración de una función *"función"* con parámetros *"x" "y": función(x, y)*. Como se puede ver, la función *"función"* recibe los parámetros *x* e *y* los cuáles a lo largo de la implementación de los algoritmos es necesario que la modificación que sufra su valor dentro de esa función, se mantenga una vez que se haya salido de ella. Esto solo es posible si esos parámetros se pasan por referencia y no por valor. En el paso por valor, lo que se le pasa a la función es únicamente el valor numérico de esa variable, mientras que en el paso por referencia se suministra la dirección de memoria en la que esa variable está alojada y que contiene su valor numérico actual. Si esa variable pasada por referencia se modifica, el valor que contiene su dirección de memoria será modificado de forma global, cosa que no sucede si se pasa por valor. En *C#* el paso por referencia se hace mediante las palabras reservadas *ref y out* pero en Java únicamente existe el paso por valor. Para solucionar esto, se han convertido las variables que necesitaban modificarse por referencia en *arrays*. Un *array* no es más que un conjunto de un tipo concreto de datos, por ejemplo, de enteros. Se ha decidido esta solución porque cuando se pasa a una función la

posición de un *array*, por ejemplo *x[0]* (en este caso la posición cero del *array*), se está pasando en realidad la posición de memoria de ese elemento del *array*, por lo que el valor de la variable quedaría modificado incluso después de haber salido de la función.

- *Arrays dinámicos:* ocurre que en *C#* es posible inicializar un *array* a un valor *null*, es decir, a lo que se podría entender como la nada, y posteriormente rellenarlo de forma dinámica, es decir sin especificarle una longitud determinada previamente al *array*. En Java, hacer esto provoca una excepción de puntero nulo que corta totalmente la ejecución de la aplicación. Esto es así porque el lenguaje interpreta que cuando haces referencia a ese *array* declarado como *null* te estas refiriendo a la nada y no puedes dar valor a la nada. Por esto, en la traducción del lenguaje ha sido necesario especificar la longitud concreta para cada *array* que se ha utilizado en la aplicación por lo que ha sido necesario en numerosas ocasiones modificar la posición de ciertas funciones que generaban los límites de esos *arrays* y ponerlas previas a la declaración de los mismos. De esta forma se declaraba un *array* distinto de *null* evitando las excepciones de puntero nulo.

6.3 Obtencion y procesado de imágenes

En este apartado se describirá el método utilizado para la selección y el procesado de las imágenes de huellas dentro de la aplicación.

Las imágenes de huellas que se proporcionaron en las bases de datos para trabajar con ellas presentaban un formato *".tiff"* ya que tras tomarlas con los lectores de huella se generaban con este formato de imagen. C# está preparado para trabajar con este formato de imágenes, sin embargo, Java no. Por ello, el primer paso para procesar las imágenes fue convertirlas a un formato más estandarizado con el que Java pudiera trabajar sin problemas, como pueden ser *".png"* o *".bmp" (mapa de bits).* Cabe destacar que todas las imágenes usadas para el procesado se convirtieron a formato *".png"* puesto que el formato JPEG puede introducir artefactos en las imágenes alterando así su calidad.

Tras esto se procedió a realizar el método de selección de imágenes el cual, para mayor comodidad para el usuario, consiste en un simple selector que se activa cuando pulsamos el botón de búsqueda (lupa) en las distintas activities de la aplicación que lo requieren. Este selector lleva al usuario directamente a la galería, permitiendo que éste escoja la imagen de huella deseada. A continuación se muestra el fragmento de código que implementa esta acción para comentarlo posteriormente.

Código del *"Listener"* para la selección de imagen:

```
load.setOnClickListener(new View.OnClickListener() {

        public void onClick(View v) {
                // TODO Auto-generated method stub
                gallery.setType("image/*");
                gallery.setAction(Intent.ACTION_GET_CONTENT);
                startActivityForResult(
                                Intent.createChooser(gallery, "Select Image"),
                                requestImg);
                        }
                });
```

Código del resultado de la selección de imagen:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
// TODO Auto-generated method stub
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == requestImg) {

                Uri selectedImgUri = data.getData();
                String[] filePathColumn = { MediaStore.Images.Media.DATA };

                Cursor cursor = getContentResolver().query(selectedImgUri,
                                filePathColumn, null, null, null);
                cursor.moveToFirst();









                int columnIndex = cursor.getColumnIndex(filePathColumn[0]);
                filePath = cursor.getString(columnIndex);
                cursor.close();
                bmp = BitmapFactory.decodeFile(filePath);
                input.setImageBitmap(bmp);

                }

        }
```

Como se puede observar en los fragmentos anteriores, el código para la selección de la imagen se divide en dos. Primero se implementa la acción que ocurrirá al pulsar el botón de cargar imagen *(load)*, esto es equivalente a establecer lo que se conoce como *Listener* al botón. La variable *gallery* (en azul ya que es una variable global) es un *Intent* que como ya se explicó se utiliza para llamar a otra activity. Las siguientes líneas implementan la llamada a la galería por parte del Intent especificando que se va a hacer como un selector (*chooser*) para escoger un elemento de la galería.

La siguiente parte del código implementa lo que sucederá una vez escogida la imagen de la galería ("Resultado de la selección de la imagen"). Como se puede ver, el primer paso es guardar los datos de la imagen en un *URI*, el cual no es más que un identificador de la dirección del objeto seleccionado que contiene todos sus datos. A continuación, los datos de la imagen contenida en el URI se guardan en un array de strings que es asociado a un *Cursor*. Un *Cursor* es una colección de filas por las que nos podemos mover de forma totalmente aleatoria. La sentencia *moveToFirst()* permite leer cualquier dato del cursor ya que este comienza posicionado antes de la primera fila. A continuación se guarda en el entero *columnIndex* el valor

numérico de la primera posición del cursor para, posteriormente, guardar su contenido (el "path" de la imagen) en el string *filePath.* Una vez asignado el contenido a *filePath*, se procede a decodificar la imagen que contiene esa dirección de memoria en la variable tipo Bitmap *bmp* pudiendo hacer cualquier cosa con ese Bitmap.

6.4 Listas dinámicas

Para realizar las listas de usuarios que aparecen en las activities de reclutamiento y verificación se ha elegido implementar estas listas dinámicas por su alta comodidad y eficiencia, más conocidas en el lenguaje Android como *ListView*. Este tipo de *view* es una de las opciones que se pueden escoger durante la creación de la interfaz gráfica en los archivos *.xml*. En la imagen de la derecha se puede observar el resultado visual de la implementación de una lista dinámica. Las principales ventajas de estas listas dinámicas se enumeran a continuación:

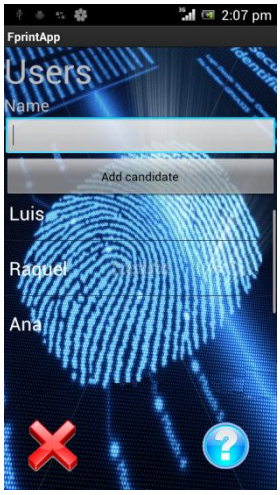

Ilustración 66: ListView.

- *Facilidad de implementación:* como se ha explicado anteriormente, los *ListView* se pueden añadir a la interfaz gráfica de la activity durante la creación de los archivos de *layout .xml.* Añadirlos es tan fácil como arrastrarlos desde el menú de *Views* que ofrece Android hasta nuestra plantilla de *layout.* Más adelante se incluirán fragmentos de código en los que se explicarán los pasos principales para implementar el *ListView* en una aplicación Android.

- *Alta eficiencia:* tras programar el *ListView* mediante unas pocas líneas de código, éste estará totalmente listo para funcionar. Su principal ventaja es que la lista se modifica de forma totalmente dinámica sin que el usuario deba preocuparse de actualizar los cambios en la lista una vez que se haya borrado un elemento o cambiado su contenido. Más adelante se verá el código que implementa la función de borrar usuarios de la lista para apreciar perfectamente el potencial del *ListView.*

- *Alta comodidad:* los *ListView* proporcionan al usuario una gran comodidad. Por ejemplo, si el número de elementos de la lista es bastante grande, muchos de los elementos no se verán en la pantalla a primera vista. Lo único que el usuario deberá hacer para acceder a ellos será arrastrar la lista con el dedo y esta se moverá hacia arriba o hacia abajo mostrando todos sus elementos. Por otro lado, si se decide borrar un elemento de la lista, ésta se actualizará automáticamente llevando todos los elementos que se encuentren en las posiciones siguientes al que se ha borrado una fila más arriba, evitando de esta forma que queden

huecos vacíos sin que el usuario tenga que preocuparse por ello.

6.4.1 Implementación del ListView mediante código

Ahora se presentará el código simplificado de cómo se implementa el *ListView* ya que posteriormente se incluirá un anexo con el código comentado. Las funciones *getIdListFile()* y *convertByteArrayToArrayList()* se explicarán con más detalle en la sección de "Cargado y guardado de archivos", ya que forman parte de estos procesos.

```
ListView recruitList;
ArrayAdapter<String> myList;
ArrayList<String> candidateList;


// Inicialización de variables
recruitList = (ListView) findViewById(R.id.lvCandidates);
candidateList = new ArrayList<String>();

// Instanciación del ArrayAdapter y asignación del mismo a nuestro
// ListView
myList = new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, candidateList);
recruitList.setAdapter(myList);

// Carga de la lista guardada en cada inicio
loadList = getIdListFile(listFilePath);
if (loadList != null) {
        byte[] inputCandidates = loadList.getBytes();
        candidateList.addAll(convertByteArrayToArrayList(inputCandidates));
        myList.notifyDataSetChanged();
} else {
        candidateList.clear();
        myList.notifyDataSetChanged();
}
```

El código comienza con la declaración de todos los elementos necesarios para la implementación del *ListView*. La variable *recruitList* que, como se puede observar, es de la clase *ListView* será la que contenga el *View* como tal. La variable *myList* es un adaptador que contendrá la información de la variable *candidateList*, que implementa la lista como un conjunto de strings y soporta las operaciones de añadir, eliminación y modificación de la lista. A continuación, se inicializan las variables. Como vemos, la sentencia *findViewById()* permite encontrar un *View* que se haya creado en el archivo de *layout (.xml)* de la activity mediante la identificación que se le haya otorgado. Después se instancia la variable *candidateList* indicando que va a ser un nuevo *ArrayList* de tipo string. Finalmente, la variable *myList* se instancia como un nuevo *ArrayAdapter* indicando la apariencia visual que tendrá la lista ("*simple_list_item_1*") y la lista que contendrá este adaptador ("*candidateList*"). Para finalizar la fase de inicialización, se asocia el adaptador a la variable *recruitList* (que contiene el *View*) mediante la sentencia *setAdapter*.

La parte final de este fragmento de código implementa la carga de la lista actualizada cada vez que se abra esta activity. Como se ha explicado antes, las funciones *getIdListFile()* y

convertByteArrayToArrayList() se detallarán en el apartado de guardado y carga de archivos. Las funciones *addAll()* y *clear()* añaden nuevos elementos a la lista y borran la lista respectivamente, mientras que la función *adapter.notifyDataSetChanged()* informa al adaptador de que la lista ha cambiado y ha de actualizarla.

6.4.2 Código para borrar elementos de la lista

En este apartado se explicará el método usado para borrar elementos (usuarios) de la lista de candidatos. Se ha elegido que se dé la opción de borrar si se realiza una pulsación prolongada sobre el elemento deseado otorgando una mayor comodidad al usuario. A continuación se incluye el fragmento de código que implementa esta función:

```
recruitList.setOnItemLongClickListener(new OnItemLongClickListener() {

// A continuación se implementará la funcionalidad para eliminar un
// item(candidato)
// de la lista tras haber realizado una pulsación prolongada sobre
// el mismo
public boolean onItemLongClick(AdapterView<?> arg0, View arg1,
        int arg2, long arg3) {
// TODO Auto-generated method stub
        final String itemToRemove = ((TextView) arg1).getText()
                                                .toString();

        // Pop up de un AlertDialog para asegurar que la pulsación no ha
        // sido por error
        AlertDialog.Builder adb = new AlertDialog.Builder(
                                                Enrolment.this);
        adb.setTitle("Remove!");
        adb.setMessage("Are you sure you want to delete "
                                                + itemToRemove);
        final int positionToRemove = arg2;
        adb.setNegativeButton("Cancel", null);
        adb.setPositiveButton("Ok", new AlertDialog.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {

                 // Se borra el item seleccionado y se actualiza la lista
                 deleteListOrItem(itemToRemove);
         candidateList.remove(positionToRemove);
                 myList.notifyDataSetChanged();

                 // Posteriormente se procede a guardar la nueva lista
                 listData = convertArrayListToByteArray(candidateList);
                 saveNewListOrItem(listData, listFilePath);

              }
        });
        adb.show();

        return false;

   }
});
```

Como se puede ver, para implementar la función de borrado de elementos se ha utilizado la función *setOnItemLongClickListener()* de la variable *recruitList* que contiene la lista, la cual nos permite realizar la acción que se desee cuando un elemento de la lista es pulsado de forma prolongada. Una vez llamada esta función, se autogenera el código de una

nueva función (*OnItemLongClick*) que tiene como parámetros el adaptador de la lista a la que se está haciendo referencia, *arg1* que guarda la información del elemento de la lista y se utiliza para guardar el nombre del candidato en el string *itemToRemove*; y un entero que contiene el valor numérico de la posición del elemento. Estos parámetros reciben su valor una vez que se ha realizado una pulsación prolongada sobre un elemento de la lista.

A continuación, y por seguridad, se ha decidido implementar un *AlertDialog* que, como se explicó anteriormente, se trata de una ventana emergente que dará al usuario la posibilidad de realizar la eliminación del elemento o cancelarla. Las siguientes líneas de código dan nombres al título del *AlertDialog*, y a los sus correspondientes botones, tanto el positivo como el negativo.

Finalmente se implementa la acción que se producirá si se pulsa el botón positivo, es decir, el que permite que se realice la eliminación del sujeto. Para realizar la eliminación, se llama a la función creada llamada *deleteListorItem()*, cuyo código se muestra a continuación:

```
private void deleteListOrItem(String listFilePath2) {
    // TODO Auto-generated method stub
    try {
          FileOutputStream removeFile = openFileOutput(listFilePath2,
                                        MODE_PRIVATE);
          removeFile.write(null);
          removeFile.close();
    } catch (Exception e) {
          e.printStackTrace();
    }

}
```

La primera línea de la función básicamente abre el fichero de texto donde se encuentra ese elemento de la lista. Se puede observar que esta función consta de un parámetro que habrá que pasarle cada vez que se le llame. Como se verá en el apartado de guardado y carga de archivos, cada sujeto dispone de un fichero de texto en el que se guardan sus datos y la lista posee otro distinto. Por ello, en este caso, el parámetro que se le ha de pasar es el string que contenga el nombre de la dirección de memoria donde se encuentra ese elemento de la lista.

Una vez abierto el fichero, las siguientes líneas lo que hacen es, primero, escribir *"null"* en el fichero para que este se vacíe totalmente y, finalmente, se cierra el fichero para que éste no se corrompa al quedarse abierto.

Cada vez que se programa una apertura de fichero, el código ha de rodearse con las cláusulas *try/catch* las cuáles nos permiten gestionar cualquier excepción que se pueda producir al intentar acceder al fichero en cuestión. De esta forma la aplicación quedará protegida ante cualquier fallo de apertura de ficheros.

6.5 Guardado y carga de archivos

En este apartado se detallarán los procesos seguidos para realizar la creación de los ficheros en los cuáles se guardan todos los datos de los sujetos incluidos en la base de datos de candidatos, así como las propias operaciones de guardado y aquéllas que permiten cargar los datos de los mismos.

Cabe destacar que, por mayor seguridad, la aplicación se ha diseñado para que cada vez que se guarde una lista o la información de huella de un candidato, se haga en la memoria interna del dispositivo. De esta forma no se puede acceder al archivo, ya que no se conoce a priori la ruta donde el dispositivo ha guardado el archivo, quedando éste mucho más protegido ante posibles amenazas que si estuviera guardado en la memoria externa.

6.5.1 Guardado de listas y candidatos

Se explicará el método de guardado de las listas de candidatos incluyendo el fragmento de código correspondiente ya que se realiza la misma operación a la hora de guardar los candidatos.

```
private void saveNewListOrItem(byte[] list, String listFilePath) {
   // TODO Auto-generated method stub
   try {
         FileOutputStream fout = openFileOutput(listFilePath, MODE_PRIVATE);
         fout.write(list);
         fout.close();
   } catch (Exception e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
   }

}
```

El código para guardar tanto listas como sujetos es bastante sencillo. Como se puede ver, la función consta de los parámetros *list* y *listFilePath* de tipo array de bytes y string respectivamente. El primero proporciona la información que contiene la lista de candidatos y el segundo la dirección de memoria donde ésta se encuentra. A continuación se realiza el mismo proceso que se describió en el apartado anterior para la función *deleteListOrItem()*. La única diferencia es que, esta vez, en lugar de escribir *"null"* en el fichero, se escribe el valor proporcionado por el parámetro *list*.

6.5.2 Carga de la lista de candidatos

Cada vez que se inicia la aplicación y se abre la activity de reclutamiento o de verificación es necesario cargar la lista de candidatos en el mismo estado en el que estaba la última vez que se modificó. Estas activities son las únicas que requieren mostrar la información de la lista de candidatos, la de reclutamiento para añadir, eliminar o modificar candidatos; y la de verificación para seleccionar el candidato de la lista cuya huella se comparará con la entrante.

Por esto, se ha diseñado una función para cargar la lista actualizada cada vez que se abra una de estas dos activities:

```
public String getIdListFile(String listFilePath2) {
    // TODO Auto-generated method stub
    String getId = null;
    try {
          BufferedReader inputIdList = new BufferedReader(
                    new InputStreamReader(openFileInput(listFilePath2)));
          getId = inputIdList.readLine().toString();
          inputIdList.close();
    } catch (Exception e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
    }
    return getId;
}
```

En este caso, la función vuelve a tener un parámetro de tipo string que contendrá la dirección de memoria donde se encuentra el fichero de la lista de candidatos. Después se crea una variable de la clase *BufferedReader* que permite leer un flujo de datos de un fichero. A continuación, se utiliza la función *readLine()* para leer el texto contenido en el fichero en el que se encuentra la lista. Esta información se guarda en una variable de tipo string para poder utilizarlo posteriormente. Finalmente, se cierra el fichero.

A continuación se incluye el fragmento de código en el que se implementa la carga de datos llamando a la anterior función:

```
loadList = getIdListFile(listFilePath);
if (loadList != null) {

    byte[] inputCandidates = loadList.getBytes();
    candidateList.addAll(convertByteArrayToArrayList(inputCandidates));
    myList.notifyDataSetChanged();
} else {
    candidateList.clear();
    myList.notifyDataSetChanged();
}
```

Como se puede observar, la variable, *loadList* recoge el string devuelto por la función *getIdListFile()*. La siguiente línea, transforma ese string en un array de bytes que, posteriormente y mediante la función *convertByteArrayToArrayList()*, se transforma al tipo *ArrayList*, del cual es el *ListView*. Una vez echa esta transformación se añade la información a la lista y se actualiza mediante el adaptador para que los cambios tengan efecto. A continuación se adjunta el código de la función *convertByteArrayToArrayList():*

```
private Collection<? extends String> convertByteArrayToArrayList(
          byte[] inputCandidates) {
    // TODO Auto-generated method stub
    ArrayList<String> ListaCandidatos = new ArrayList<String>();
    ByteArrayInputStream inByteArray = new ByteArrayInputStream(
              inputCandidates);
    DataInputStream input = new DataInputStream(inByteArray);
    try {
          while (input.available() > 0) {
                String candidate = input.readUTF();
                ListaCandidatos.add(candidate);
          }
    } catch (Exception e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
    }
    return ListaCandidatos;
}
```

Si analizamos el tipo de la función, se observa que ésta devolverá una variable de tipo *Collection* que contiene un tipo string, la cuál es la que se necesita para rellenar el *ListView*.

Primero se crea la variable auxiliar *ListaCandidatos* de tipo *ArrayList* que será la que la función devolverá. Después se recoge la variable de tipo byte array que se pasa como parámetro en la variable *inByteArray*. La cual permite leer un flujo de datos de la variable pasada como parámetro.

En el bloque *try/catch* se implementa la lectura de líneas contenidas en este array de bytes para almacenarlas en el string *candidate* y añadirlas en la variable auxiliar *ListaCandidatos* mediante su función *add()*. Esta operación se hace para todas las líneas del array de bytes a través del bucle *while*.

Finalmente se devuelve la variable *ListaCandidatos* rellena que contiene la información de todos los sujetos almacenados en la lista.

6.6 Envío de datos entre activities

Para detallar este apartado se pondrá como ejemplo la activity de verificación, la cual parte de la lista de candidatos y, una vez que se pulsa sobre uno de los usuarios, se abrirá otra activity en la que se realizará la propia verificación. Para que esto sea posible, es necesario que la activity padre (en la que se encuentra la lista de candidatos) se comunique con la hija pasándole los datos del sujeto que se ha seleccionado. A continuación se incluye el fragmento de código en el que la activity padre manda los datos de usuario a la hija. Este fragmento de código se encuentra implementado en la activity padre, que en el proyecto tiene por nombre

Verification.java:

```
verifList.setOnItemClickListener(new OnItemClickListener() {

    public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
              long arg3) {
          // TODO Auto-generated method stub

          final String itemID = ((TextView) arg1).getText().toString();
          final int itemPos = arg2;
          Intent goVerif = new Intent(Verification.this,
                    VerifyActivity.class);
          goVerif.putExtra("candidateID", itemID);
          goVerif.putExtra("candidatePos", itemPos);
          startActivityForResult(goVerif, 0);

    }

});
```

Como se puede ver, el fragmento comienza dentro de la función *OnClickListener* que hace referencia a lo que ocurrirá cuando se realice una pulsación sobre uno de los elementos de la lista. Esta vez es una pulsación corta a diferencia de como ocurría en la activity de reclutamiento a la hora de borrar sujetos.

Como ya se explicó, los parámetros de esta función *Listener* incluyen, entre otros, el nombre del elemento y su posición en la lista. Las primeras líneas dentro de la función, guardan estos datos en el string *itemID* y en el entero *itemPos* respectivamente. A continuación, se declara un *Intent* llamado *goVerif* que, como se ve arriba, se utiliza para abrir la activity *VerifyActivity* que es en la que se realizará la verificación como tal.

Después se utiliza la función *putExtra* incluida en la clase *Intent* para que la posición y el nombre del sujeto sean pasados a la vez que se abre la nueva activity mediante el *Intent goVerif*. Esta función consta de dos parámetros, el primero es una *KEY* que no es más que una cadena de caracteres que se utiliza como identificador de la variable que se está incluyendo como extra. El segundo parámetro es la propia variable que se quiere pasar a la nueva activity.

Finalmente, se llama a la función *startActivityForResult()* pasándole como parámetro el *Intent goVerif* con todos los datos incluidos en él. Esto hará que se abra la nueva activity a la vez que se le pasan los datos.

Ahora se verá el fragmento de código en el que se recogen los datos que se han pasado a la nueva activity. Este código se ha de implementar dentro del archivo *.java* de la activity que se pretende abrir, en este caso la activity *VerifyActivity.java*:

```
private void collectItemData() {
    // TODO Auto-generated method stub

    // Recogida de datos
    Bundle getData = getIntent().getExtras();
    userID = getData.getString("candidateID");
    userPos = getData.getInt("candidatePos");
    user.setText("User: " + userID);
    user.setTextColor(Color.MAGENTA);
```

```
}
```

Para realizar la recogida de datos se ha creado una función llamada *collectItemData()* como se puede apreciar en el fragmento de código. Esta función se basa en el uso de la clase *Bundle*. Esta palabra se puede traducir del inglés como paquete o algo similar, y es básicamente lo que esta clase es. *Bundle* es una especie de recipiente que permite recoger todos los extras que se han pasado asociados al *Intent* que ha invocado a la activity. Si se ve el código, la primera línea de la función corresponde con la creación de una variable tipo *Bundle* llamada *getData* y se está llamando a la función *getIntent().getExtras(),* las cuales permiten que la variable se cree conteniendo en ese momento todos los extras del *Intent.*

Una vez hecho esto, solo queda recoger los extras, que como se vio, son la posición del elemento de la lista y su nombre. Para ello se crean dos variables, una tipo string y un entero que recogerán ambos datos de la variable de tipo *Bundle*. De esta forma, esos datos quedan totalmente accesibles para utilizarlos dentro de la activity de la forma que se desee.

6.7 Detección por pasos

En este apartado se detallará el código implementado para realizar la activity de detección de minucias paso a paso. Para ello, se adjunta primero el fragmento de código que implementa esta función para proceder a comentarlo posteriormente:

```
step.setOnClickListener(new View.OnClickListener() {

    public void onClick(View v) {
       // TODO Auto-generated method stub

       if (filePathS != null) {
           switch (contador) {

        case 0:

           Bitmap mapa = bmpS;
           inputS.setImageBitmap(mapa);
           step.setText("Next");


        // Procedemos a obtener el array de minucias y sus
          // caraterísticas
        // Comenzamos la cuenta del tiempo de proceso
            tStartS = System.nanoTime();

            ***Detección de las minucias
        // Fin de la cuenta de tiempo
            tEndS = System.nanoTime();
            tOffsetS = tEndS - tStartS;
            bz_gbls.tOverallTime = tOffsetS / 1000000;

            minutiaSave[0] = ominutiae[0];

        // Se dibujan los círculos rojos

            BitmapFactory.Options bfo = new BitmapFactory.Options();
            bfo.inSampleSize = 5;
```

```
        overlay = Bitmap.createBitmap(mapa.getWidth(),
        mapa.getHeight(), mapa.getConfig());
        canvas = new Canvas(overlay);
        paint = new Paint();
        paint.setColor(Color.RED);
        paint.setStyle(Style.STROKE);
        paint.setStrokeWidth((float) 2.5);
        canvas.drawBitmap(mapa, new Matrix(), null);

        for (int d = 0; d < minutiaSave[0].num; d++) {
            int m_x = minutiaSave[0].list[d].x;
              int m_y = minutiaSave[0].list[d].y;
              canvas.drawCircle(m_x, m_y, 10, paint);
        }

        inputS.setImageBitmap(bz_gbls.binar);
        contador++;

        // Procedemos a rellenar los campos que contendrán la
        // información de tiempo
        // y descripción del proceso

        pth1S.setText(binarDescription);
        pth1S.setTextColor(Color.WHITE);

      processTimer.setText("Binarization time =
                  + bz_gbls.tBinarMillis + " ms");
      processTimer.setTextColor(Color.GREEN);

       break;

    case 1:
      inputS.setImageBitmap(overlay);
        Context context = getApplicationContext();
      CharSequence text = ".xyt file generated";
        int duration = Toast.LENGTH_SHORT;
        Toast toastMin = Toast.makeText(context, text, duration);
        toastMin.show();
        step.setText("Finish");
        contador++;

       // Procedemos a rellenar los campos que contendrán la
       // información de tiempo
       // y descripción del proceso

        pth1S.setText(detectDescription);
        pth1S.setTextColor(Color.WHITE);

        processTimer.setText("Detection time = "+ bz_gbls.tDetectMillis +
                    ");

      processTimer.setTextColor(Color.GREEN);

        break;

    case 2:

       pth1S.setText("OVERALL PROCESS");
       pth1S.setTextColor(Color.WHITE);

       processTimer.setText("Overall time = " + bz_gbls.tOverallTime
                                                  + " ms");
     processTimer.setTextColor(Color.GREEN);
     step.setText("Exit without save");
       contador++;

       break;

  case 3:

     finish();

       break;
```

Como se puede observar en el código, el elemento principal que ha permitido el desarrollo de la detección por pasos es un contador. Este contador es la variable entera *contador*. El código se encuentra implementado dentro del *Listener* del botón de detección, ya que será la pulsación de este la que permitirá ir avanzando por los pasos.

La forma de realizarlo ha sido utilizando una sentencia *switch/case* cuyo funcionamiento consiste en evaluar una variable, en este caso *contador*, y dependiendo del valor que posea en cada momento se realiza una acción u otra.

Como se puede ver en el fragmento de código, se han implementado acciones para cuatro valores distintos de *contador*. Para simplificar el código se ha decidido realizar la detección de minucias al completo para el valor '0' de *contador* y se muestran distintos resultados a medida que su valor se va incrementando.

Se puede apreciar que el contador se incrementa en cada una de las cuatro acciones mediante la sentencia "++" que incrementa su valor en una unidad. Además, cada vez que el valor se incrementa, significa que se ha pasado a un nuevo paso, por lo que se cambia el texto que muestra el botón hasta finalizar con el texto "Exit without save".

Se puede observar que la cuenta de tiempo se lleva en todo momento a través de la función *System.nanotime()*. Esta es una herramienta del sistema que proporciona Java, la cual muestra una cuenta de tiempo en nanosegundos cada vez que se le llama. Por ello para calcular el tiempo que requiere un proceso determinado se llama a esta función antes y después del proceso. Después se calcula el offset mediante la diferencia entre ambos tiempos y se convierte a milisegundos dividiendo entre 1.000.000 para tener una unidad de tiempo más manejable.

Otro aspecto destacable de la implementación de esta función es el código del paso en el que se muestran todas las minucias detectadas rodeadas por círculos rojos superpuestos en la imagen de huella original en escala de grises. Esto se ha conseguido utilizando la clase *Canvas*, la cual permite crear una capa sobre la imagen original en la que se dibujaran los círculos que rodearán las minucias. Tras crear el *Canvas* y el Bitmap auxiliar que contendrá el dibujo, se procede a especificar el estilo, el color y el ancho de línea de lo que vayamos a dibujar en nuestra capa auxiliar. Posteriormente se ha creado un bucle *for* que recorre todas las minucias detectadas y guardadas en el archivo de minucias, extrayendo las información de sus coordenadas *x* e *y* para así poder establecer el centro de los círculos. Después se dibuja un círculo en cada uno de los centros a través de la función de la clase *Paint drawCircle()*.

Finalmente, para poder implementar la función de guardar sin salir, simplemente se llama a la función *finish()* que cierra la activity actual. Esta función se llama cuando se alcanza el último paso de la detección, que coincide cuando el contador alcanza el valor numérico 3.

6.8 Seguridad

A lo largo de las distintas activities se han incluido una serie de medidas de seguridad para prevenir cualquier tipo de comportamiento inesperado para el código por parte del usuario. Estas situaciones son aquellas que no siguen el proceso esperado por el código y provocan que la aplicación falle y se fuerce su cierre. Estas situaciones se producen al intentar pulsar algún botón que requiera la existencia de una imagen para ejecutar el código y esa imagen no se encuentre cargada.

Esto ocurre en las activities de detección de minucias, verificación e identificación cuando se pulsa el botón de detectar, verificar o identificar respectivamente, sin que haya ninguna imagen cargada. También ocurre en la activity de detección si se intenta pulsar el botón de guardar archivo sin ninguna imagen cargada. Si no se hubiera implementado ninguna función para esta situación, la aplicación fallaría si ocurriera y se vería forzada a cerrar. Por ello, a continuación se detalla el fragmento de código de esta protección:

```
verify.setOnClickListener(new View.OnClickListener() {

    public void onClick(View v) {
       // TODO Auto-generated method stub

       if (filePath != null) {

                Código para la verificación…

       } else {
             try {
                   CharSequence noV = "Load an image first";
                   Toast noVerif = Toast.makeText(VerifyActivity.this,
                             noV, Toast.LENGTH_LONG);
                   noVerif.show();
                   } catch (Exception e) {
                       e.printStackTrace();
                   }
        }
    }
});
```

Para ilustrar la implementación del este método de seguridad, se ha elegido exponer el código del mismo que se encuentra en la activity de verificación *(VerifyActivity.java)*. Se puede comprobar que el código de seguridad está dentro del *Listener* para el botón de verificación.

Lo primero que se comprueba tras pulsar el botón es si hay alguna imagen cargada. Para ello, se ha elaborado una sentencia *if* en la que se comprueba que el string *filePath* sea distinto de *null*, es decir, que se encuentre relleno por la dirección de memoria de alguna imagen.

Si es así, la aplicación procede a la verificación de la imagen con el usuario seleccionado. Si no, se aplicaría el método de protección. Este método consiste simplemente en mostrar un *Toast* por pantalla. Como ya se explicó un *Toast* es un mensaje emergente que se superpone a cualquier activity que haya abierta en ese momento. Para crear el *Toast*, se rellena la cadena de caracteres *noV* con el mensaje que se quiere mostrar por pantalla. A continuación

se crea la variable de la clase *Toast* especificando que el texto que ha de mostrar es el contenido de la variable *noV* y la duración que se desea que el mensaje permanezca en la pantalla, a elegir entre *LENGTH_SHORT* y *LENGTH_LONG*. El primer parámetro (*VerifyActivity.this*) hace referencia al contexto en el que se quiere mostrar el *Toast*, en este caso la propia activity de verificación.

Una vez que se han especificado todos los parámetros del *Toast*, únicamente queda llamar a la función *show()* de la clase *Toast*. De esta forma, cuando se pulse el botón y la aplicación compruebe que no existe ninguna imagen cargada, se mostrará un mensaje en la pantalla recordando que es necesario cargar una imagen primero para continuar con la verificación.

Este sistema de protección se ha implementado para el resto de activities en las que es posible que ocurra esta situación. Como ya se comentó antes, estas activities son las de detección de minucias, verificación e identificación.

7. Pruebas

En este capítulo se incluirán los resultados obtenidos al hacer pruebas con diferentes imágenes de huella dactilar existentes en las bases de datos proporcionadas para la realización de este proyecto.

Se dividirá este capítulo en dos apartados en lo que se incluirán sendas tablas que incluirán los resultados de una prueba de tiempos de detección y de puntuación y tiempos en la verificación respectivamente.

En la tabla de resultados de la verificación se han utilizado huellas de distintos individuos y, para un mismo individuo, se han incluido varias huellas de su mismo dedo tomadas en tiempos diferentes para así poder sacar una gráfica de resultados de falso rechazo y falsa aceptación.

7.1 Resultados tiempos de detección

Este apartado de resultados incluye una tabla en la que se ha efectuado el tiempo que lleva realizar distintos procesos para la detección de las minucias. Para aclarar lo reflejado en la tabla, se va a recordar los procesos seguidos para la detección de las minucias:

1. Decodificación de la imagen en escala de grises.
2. Generación de los mapas para procesar la imagen.
3. Binarización de la imagen.
4. Detección de las minucias.
5. Eliminación de las falsas minucias.

Para elaborar esta tabla se han tenido en cuenta los procesos de la detección por pasos: la binarización, la detección de minucias y el proceso global. La columna de "Tiempo total" incluye el tiempo desde que se decodifica la imagen en escala de grises hasta que se genera el archivo .xyt.

Ilustración 67: Tiempos de la detección de minucias

Huella candidato	Tiempo binarización	Tiempo detección minucias	Tiempo total
1_1	6 ms	1055 ms	3094 ms
1_2	6 ms	993 ms	2985 ms
1_3	5ms	903 ms	2691 ms
1_4	6ms	946 ms	2591 ms
2_1	6 ms	1018 ms	3015 ms

Huella candidato	Tiempo binarización	Tiempo detección minucias	Tiempo total
2_2	6 ms	958 ms	2840 ms
2_3	6 ms	981 ms	2944 ms
2_4	6 ms	918 ms	2756 ms
3_1	6 ms	1071 ms	2974 ms
3_2	6 ms	1076 ms	2951 ms
3_3	6 ms	990 ms	3066 ms
3_4	6 ms	974 ms	2891 ms
4_1	7 ms	1024 ms	3024 ms
4_2	5 ms	709 ms	2036 ms
4_3	6 ms	941 ms	2603 ms
4_4	6 ms	902 ms	2665 ms
5_1	6 ms	922 ms	2882 ms
5_2	6 ms	923 ms	3120 ms
5_3	6 ms	1027 ms	2920 ms
5_4	6 ms	1006 ms	2879 ms
6_1	6 ms	933 ms	2612 ms
6_2	6 ms	917 ms	2707 ms
6_3	6 ms	926 ms	2648 ms
6_4	6 ms	963 ms	2786 ms
7_1	6 ms	900 ms	2796 ms
7_2	6 ms	897 ms	2585 ms
7_3	6 ms	1034 ms	2933 ms
7_4	6 ms	1092 ms	3226 ms
8_1	6 ms	1458 ms	3610 ms
8_2	10 ms	1874 ms	4203 ms
8_3	6 ms	1079 ms	2902 ms
8_4	5 ms	1002 ms	2859 ms

Las huellas de candidatos se encuentran ordenadas por individuos. El primer número indica el número del individuo mientras que el segundo número indica el número de huella perteneciente a él mismo. Se pondrá un ejemplo para aclarar esta explicación: la huella 1_1 indica la huella número 1 del candidato número 1 y la huella 1_2 indica la huella número 2 del candidato número 1, es decir, estas dos huellas pertenecen al mismo sujeto mientras que, por ejemplo, la huella 2_1 es la huella número 1 del candidato número 2, por lo que sería una huella de un usuario totalmente distinto al número 1.

El tiempo de binarización se corresponde con el tiempo que la aplicación tarda en binarizar la imagen, es decir, en pasar los píxeles pertenecientes a crestas a color negro y los pertenecientes a valles a color blanco. Como se puede comprobar, este proceso requiere un tiempo mínimo y es prácticamente el mismo independientemente de la huella que se esté procesando.

El tiempo de detección de huellas es el tiempo invertido por la aplicación en realizar los

procesos para aislar las minucias, eliminar las falsas y generar el archivo *.xyt*, este es uno de los procesos más largos y por ello supone casi un 50% del tiempo total de procesado.

El tiempo total engloba también el tiempo destinado a la decodificación de la imagen en escala de grises para guardar todos sus píxeles en arrays, es decir, el tiempo que lleva leer todos los píxeles de la imagen y extraer la información que contienen. Este es el proceso más largo a la hora de procesar la imagen.

Tras realizar una media de los tiempos obtenidos, se obtienen los siguientes resultados, los cuales suponen tiempos muy aceptables para estar realizando el procesado de las imágenes a través de un dispositivo móvil con las limitaciones que ello implica frente al procesado por ordenador:

- *Tiempo medio de binarización:* 6.06 ms
- *Tiempo medio de detección:* 1012.87 ms
- *Tiempo medio global:* 2899.81 ms

7.2 Resultados verificación

Anteriormente se realizó la explicación de la nomenclatura de los candidatos por lo que no es necesario repetirla en este apartado. Como se puede comprobar, para obtener los resultados de verificación se ha combinado la comparación entre huellas tanto del mismo individuo como de diferentes. De esta forma se pueden obtener resultados de falso rechazo y falsa aceptación.

Ilustración 68: Tiempos comparación huellas

Candidato A	Candidato B	Tiempo verificación	Resultado verificación	¿Mismo sujeto?
1_1	1_1	9257 ms	509	Sí
1_1	2_1	817 ms	9	No
1_1	3_1	1043 ms	13	No
1_1	4_1	1036 ms	10	No
2_1	2_1	2140 ms	293	Sí
2_1	3_1	2024 ms	23	No
2_1	4_1	859 ms	7	No
3_1	3_1	16382 ms	474	Sí
3_1	4_1	1517 ms	18	No
4_1	4_1	10875 ms	393	Sí
1_2	1_2	2565 ms	476	Sí
1_2	1_1	1915 ms	160	Sí

Candidato A	Candidato B	Tiempo verificación	Resultado verificación	¿Mismo sujeto?
1_2	2_1	944 ms	16	No
1_2	3_1	1150 ms	15	No
1_2	4_1	566 ms	16	No
2_2	1_2	966 ms	11	No
2_2	1_1	888 ms	8	No
2_2	2_1	910 ms	57	Sí
2_2	3_1	384 ms	9	No
2_2	4_1	282 ms	4	No
3_2	3_2	7575 ms	488	Sí
3_2	1_2	1035 ms	10	No
3_2	2_2	836 ms	8	No
3_2	1_1	1247 ms	10	No
3_2	2_1	1202 ms	22	No
3_2	3_1	5939 ms	233	Sí
3_2	4_1	1077 ms	14	No
4_2	4_2	4910 ms	446	Sí
4_2	1_2	1108 ms	19	No
4_2	2_2	784 ms	8	No
4_2	3_2	1225 ms	25	No
4_2	1_1	938 ms	15	No
4_2	2_1	817 ms	10	No
4_2	3_1	2994 ms	22	No
4_2	4_1	14131 ms	114	Sí
1_3	1_3	8354 ms	481	Sí
1_3	1_2	902 ms	8	Sí
1_3	2_2	267 ms	6	No
1_3	3_2	914 ms	9	No
1_3	4_2	832 ms	6	No
1_3	1_1	321 ms	6	Sí
1_3	2_1	243 ms	7	No
1_3	3_1	512 ms	7	No
1_3	4_1	357 ms	5	No
2_3	2_3	1102 ms	328	Sí
2_3	1_3	799 ms	7	No
2_3	1_2	889 ms	8	No
2_3	2_2	2864 ms	168	Sí
2_3	3_2	1027 ms	12	No
2_3	4_2	837 ms	12	No
2_3	1_1	332 ms	7	No
2_3	2_1	344 ms	27	Sí
2_3	3_1	347 ms	11	No
2_3	4_1	250 ms	7	No
3_3	3_3	2079 ms	422	Sí

Candidato A	Candidato B	Tiempo verificación	Resultado verificación	¿Mismo sujeto?
3_3	1_3	789 ms	6	No
3_3	2_3	981 ms	13	No
3_3	1_2	947 ms	10	No
3_3	2_2	832 ms	14	No
3_3	3_2	3019 ms	21	Sí
3_3	4_2	963 ms	12	No
3_3	1_1	335 ms	11	No
3_3	2_1	283 ms	13	No
3_3	3_1	1092 ms	23	Sí
3_3	4_1	301 ms	8	No
4_3	4_3	14357 ms	442	Sí
4_3	1_3	340 ms	7	No
4_3	2_3	319 ms	11	No
4_3	3_3	1427 ms	18	No
4_3	1_2	979 ms	8	No
4_3	2_2	773 ms	5	No
4_3	3_2	3334 ms	29	No
4_3	4_2	35113 ms	-1	Sí
4_3	1_1	890 ms	17	No
4_3	2_1	468 ms	16	No
4_3	3_1	5335 ms	29	No
4_3	4_1	5346 ms	139	Sí

Como se ve en la tabla, por norma general la comparación de huellas del mismo individuo da resultados superiores a 170 puntos, pero esto no siempre ocurre. Algunas comparaciones entre las mismas huellas tomadas en distintas ocasiones llegan a dar puntuaciones iguales o incluso inferiores a aquéllas obtenidas en la comparación de huellas totalmente diferentes, de distintos individuos.

En cuanto al tiempo de verificación, se puede observar que es superior en la comparación de huellas con alta semejanza, esto es así porque la cantidad de minucias similares es mayor y, por tanto, la aplicación ha de utilizar más recursos para generar la puntuación resultante.

Si se analiza el promedio del tiempo de verificación el resultado es 2651.08, un resultado muy aceptable. Es necesario aclarar que aquellas huellas que son totalmente diferentes requieren un tiempo ínfimo para la comparación puesto que el algoritmo en este caso es muy corto al no existir a penas concordancias entre las minucias.

Una vez tomados los datos de la tabla anterior, se ha procedido a realizar la gráfica en la que se representa la tasa de falsa aceptación (FAR), falso rechazo (FRR) y el EER. El EER es el punto en el que FAR y FRR coinciden. Para ello se ha calculado la distribución normal por un lado de los resultados de comparación entre huellas de los mismos individuos, y por otro de los de distintos individuos.

Con el cálculo de la distribución normal de cada resultado a través del promedio y la desviación estándar, se han realizado las gráficas gaussianas que determinan los valores para las tasas anteriormente mencionadas. A continuación se muestra esa gráfica.

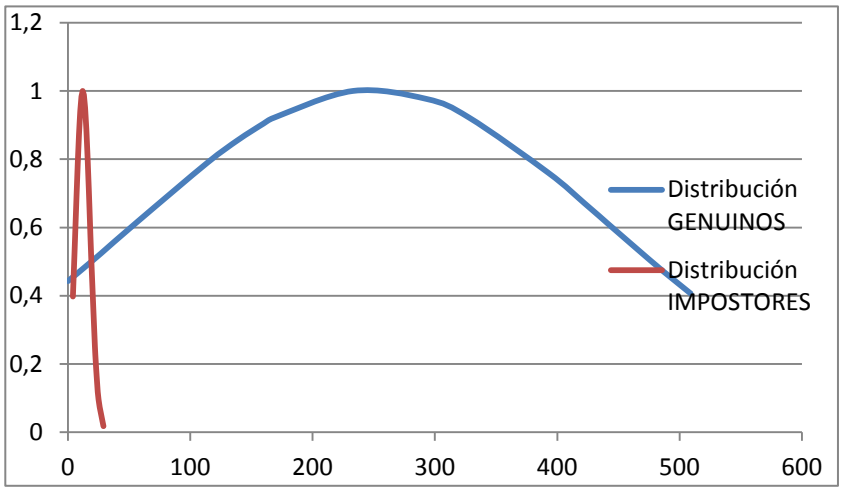

Ilustración 69: Distribución normal resultados verificación

Si se observa la anterior gráfica, se pueden apreciar dos gráficas correspondientes a funciones gaussianas. La roja representa la distribución normal de los resultados obtenidos de las comparaciones de huellas de distintos candidatos, mientras que la azul es lo mismo aplicado a las comparaciones entre los mismos inviduos.

Como se puede observar, el punto en el que ambas gráficas se cortan pertenece a un valor de resultado de 29 y es equivalente al EER. Este es el umbral en el que el número de falsas aceptaciones (verificar a un individuo distinto al de la huella) y falsos rechazos (rechazar al individuo propietario de la huella) coincide.

Si se aumentara el valor de este resultado, el sistema sería más restrictivo ya que aumentaría el índice de falso rechazo, mientras que si se disminuye el sistema sería más permisivo incrementando el índice de falsas aceptaciones.

En órden de establecer un límite númerico equitativo para los resultados, se ha decidido incrementar ligeramente este valor a 35 para establecer el umbral de verificación. De esta forma el sistema es levemente más restrictivo pero sin aumentar de forma desmedida el índice de falso rechazo.

7.3 Comparación con aplicación de C#

A continuación se van a comparar las medias de tiempo obtenidas en el dispositivo Android con las obtenidas implementando el algoritmo en lenguaje C# en un PC. Primeramente se mostrará una tabla que recoge las medias obtenidas en el PC. En esta tabla se representa el tiempo medio para distintas bases de datos de imágenes de huella (DB) medidas con distintos lectores:

Ilustración 70: Tiempos medios de detección en C#

	Tiempo medio (ms)	Desviación (ms)	Numero de minucias medio obtenido	Desviación en el número de minucias
DB1	115.45	17.59	38.79	11.3
DB2	129.86	13.43	54.44	16.59
DB3	373.55	54.03	127.83	34.15
DB4	77.75	11.03	34.63	8.3
Media	174.15	24.02	63.92	17.59

Tras comparar los resultados anteriores con los obtenidos a través de la aplicación FprintApp en dispositivo Android se obtiene la siguiente tabla que muestra el porcentaje de aumento de tiempo:

Ilustración 71: Aumento de tiempo entre aplicaciones

Tiempo medio en C# (ms)	Tiempo medio en Android (ms)	Porcentaje aumento tiempo (%)
174.15	2899.81	1665.12

Como se puede comprobar, el tiempo dedicado por el dispositivo Android para realizar todos los cálculos necesarios es significativamente superior al empleado en un PC. Esto es debido a que la capacidad de cálculo de sendos microprocesadores no es comparable, siendo mucho más potente la del PC. Sin embargo, el tiempo usado por el dispositivo Android es aceptable y perfectamente válido para aplicarlo en un ámbito comercial.

8. Conclusiónes y líneas futuras

Finalmente se puede decir que se ha realizado un proyecto interesante que proporciona una gran funcionalidad a la hora de identificar personas mediante huella dactilar. Actualmente, la implantación de esta aplicación en los dispositivos móviles tendría un uso limitado debido a la carencia del hardware necesario para tomar imágenes latentes de huellas, ya que FprintApp requiere imágenes de huellas digitalizadas para su correcto funcionamiento.

Sin embargo, esta aplicación es el preámbulo para las siguientes generaciones de los sistemas de identificación. Esto es así debido a que las grandes empresas de telefonía cada vez intentan aplicar más de lleno métodos de identificación biométrica en sus dispositivos. Es el caso de los smartphones de última generación, los cuáles empiezan a implementar la función del reconocimiento facial mediante la cámara. Una posible evolución de esto sería la implantación en los dispositivos de una cámara capacitada para tomar imágenes de huellas válidas.

Esto permitiría un uso total de las funcionalidades de FprintApp pudiendo utilizar el software para distintos fines tales como desbloqueo de carpetas o de la propia pantalla, uso de la imagen de la huella para la realización de todo tipo de operaciones, etc…

A nivel personal, cabe destacar que la realización de este proyecto a supuesto para el autor un esfuerzo altamente satisfactorio, ya que partiendo desde cero, se ha podido sumergir en la programación del lenguaje Java. Un lenguaje además en pleno auge y muy presente en numerosos dispositivos y aplicaciones de uso cotidiano, tanto para PC como para dispositivos móviles.

El aprendizaje de este lenguaje a suscitado al autor fuertes deseos de continuar trabajando con él y desarrollar software útil y que pueda servir de ayuda y referencia para cualquier usuario que pueda necesitarlo.

En cuanto a las futuras líneas que podría seguir el proyecto, lo más interesante sería proceder con una mejora y optimización del código para adecuarlo a un ámbito más comercial. También se debería estudiar la posibilidad de una modificación de hardware en los dispositivos convencionales añadiendo algún lector de huella dactilar portátil que no hiciera al dispositivo incómodo a la hora de transportar.

Tras esta modificación se estudiaría mejorar el código para poder programar la comunicación con el nuevo lector incorporado. Esto podría traducirse en añadir nuevo código a las bibliotecas ya proporcionadas por Android y publicarlo para que la comunidad pudiera añadir mejoras y conseguir así el código óptimo.

De esta forma se podría conseguir una herramienta muy útil en el día a día de cada persona pudiendo ayudar en cualquier tipo de operación eliminando la necesidad de recordar contraseñas y aumentando la seguridad de los datos personales.

REFERENCIAS

[1] Android developer. http://developer.android.com

[2] Canal de Youtube de "thenewboston", Android Tutorial. http://www.youtube.com/course?list=EC2F07DBCDCC01493A

[3] DCSAppMobiles. Lista dinámica. http://dcsappmobiles.wikispaces.com/Android+Tutorial+-+Lista+Dinamica

[4] Androides y Pingüinos. Listview y ArrayAdapter. http://miguelangellv.wordpress.com/2012/05/05/listview-y-arrayadapter-en-android/

[5] Stackoverflow. Foro de programación. http://stackoverflow.com/

[6] Sistemas biométricos. Matching de huellas dactilares. http://www2.ing.puc.cl/~iing/ed429/sistemas_biometricos.htm

[7] Wikipedia. Biometría. http://es.wikipedia.org/wiki/Biometr%C3%ADa

[8] Wikipedia. Identificación de personas. http://es.wikipedia.org/wiki/Identificaci%C3%B3n_de_personas

[9] Wikipedia. Android http://es.wikipedia.org/wiki/Android

[10] http://www.net.com.mx. Venta de artículos diversos

[11] NIST. Algoritmos MINDTCT y BOZORTH3. http://www.nist.gov/itl/iad/ig/nbis.cfm

[12] Marino Tapiador Mateos, Juan A.Sigüenza Pizarro - "Tecnologías biométricas aplicadas a la seguridad" - editorial Ra-Ma, 2005.

[13] Aitor Mendaza Ormaza – "Presentación: Introducción SO Android" – GrupoUniversitario de Tecnologías de identificación, Carlos III de Madrid.

[14] Aitor Mendaza Ormaza – "Presentación: Introducción Programación Android" –Grupo Universitario de Tecnologías de identificación, Carlos III de Madrid.

[15] Raúl Sánchez Reíllo – "Verificación automática de personas mediante huella dactilar" – Ágora SIC (Volumen 3), Grupo Universitario de Tarjeta Inteligente, Dpto. Tecnología Electrónica – Grupo de Microelectrónica

[16] Eugenio Hernández Martínez – Presentación "Fingerprint algorithm & BSP development" – GUTI

[17] National Institute of Standards and Technology – "User's Guide to Export ControlledDistribution of NIST Biometric Image Software (NBIS-EC)" – Manual de referencia para los algoritmos de detección y "matching"

[18] National Institute of Standards and Technology – "User's Guide to
NIST Biometric Image Software (NBIS)" – Manual de referencia para los algoritmos de detección y "matching"

[19] http://www2.ing.puc.cl Descripción sobre los sistemas biométricos

ANEXO I: PRESUPUESTO Y PLANIFICACIÓN

A continuación se presenta un desglose del tiempo de realización del proyecto. Se ha optado por realizarlo en distintas fases dada la compleja elaboración requerida para el desarrollo de la aplicación:

Fase 1: Documentación inicial

I. Estudio de la plataforma Android y del entorno de desarrollo Eclipse (25 horas)
II. Preparación de las herramientas de trabajo (2 horas)
III. Búsqueda y realización de tutoriales y aplicaciones sencillas. (40 horas)
IV. Asistencia a charlas y presentaciones sobre Android (16 horas)

Fase 2: Desarrollo de la aplicación

I. Actividad principal (200 horas)
II. Actividades secundarias (80 horas)
III. Interconexión de todas las actividades (30 horas)
IV. Apartado gráfico (20 horas)

Fase 3: Pruebas en un dispositivo real

I. Pruebas sobre Sony Ericsson Xperia Arc (15 horas)
II. Corrección y depuración (30 horas)

Fase 4: Elaboración de la memoria

I. Redacción de la memoria (85 horas)
II. Corrección (10 horas)

Ilustración 72: Desglose de horas

FASES	HORAS EMPLEADAS
Documentación inicial	83
Desarrollo de la aplicación	330
Pruebas en un dispositivo real	45
Elaboración de la memoria	95
TOTAL	**553**

Ya se disponía previamente de todo el material utilizado para la realización de este proyecto antes de empezar con el mismo. Por ello, para estimar un coste de cada material empleado, se expondrá la fracción equivalente de la amortización según el tiempo dedicado a la realización del proyecto en base a la vida útil de cada material.

Ilustración 73: Tabla costes materiales

CONCEPTO	PRECIO (€)
Ordenador Toshiba Satellite A500	98
Samsung Galaxy Ace	22.75
Cable USB	2
TOTAL	**122.75**

COSTES DE PERSONAL

Para la realización de este trabajo, ha sido necesaria la presencia de un jefe de proyecto y un ingeniero.

Ilustración 74: Tabla costes de personal

OCUPACIÓN	HORAS	PRECIO/HORA	IMPORTE (€)
Jefe de proyecto	25	50	1250
Ingeniero	553	30	16590
TOTAL	**300**		**17840**

COSTES TOTALES

Ilustración 75: Tabla coste total

CONCEPTO	PRECIO (€)
Costes de materiales	122.75
Costes de personal	17840
Costes indirectos (20%)	3592.55
Subtotal	21555.3
IVA (21%)	4526.613
TOTAL	**26081.91**

El coste total del proyecto es de *veintiséis mil seiscientos ochenta y un euros con noventa y un céntimos.*

Leganés, 6 de Febrero de 2013

ANEXO II: Código comentado

En este capítulo se anexará el código de las activities implementadas comentado. De esta forma se podrán apreciar con total claridad los métodos seguidos para la realización de la aplicación.

Por motivos de privacidad, no se incluirá el código relacionado con la aplicación de los algoritmos tanto para la detección de minucias como para la decodificación de la imagen y la comparación de huellas. Tampoco se incluirá la importación de bibliotecas de Java puesto que es un paso que se sobreentiende, ya que es necesario para el correcto funcionamiento del código.

❖ **Menú principal:** *ListMenu.java*

```
public class ListMenu extends ListActivity {


	// Se crea un array de strings que contendrá todas las opciones que
	//permite el menú
	String classes[] = { "Enrolment", "Verification", "Identification" };

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		// TODO Auto-generated method stub

		//Se crea un adaptador para convertir el menú en una lista
		//dinámica
		super.onCreate(savedInstanceState);
		setListAdapter(new ArrayAdapter<String>(ListMenu.this,
				android.R.layout.simple_list_item_1, classes));
	}

	@Override
	protected void onListItemClick(ListView l, View v, int position, long id) {
		// TODO Auto-generated method stub
		super.onListItemClick(l, v, position, id);

		//Una vez que se ha pulsado una de las opciones del menú, ésta
		//será identificada mediante el argumento "position"
		String cheese = classes[position];
		Class menuClass;
		try {
			//Se crea el string que abrirá la nueva activity dependiendo
			//de la opción que se haya pulsado
			menuClass = Class.forName("es.jose.fpapp." + cheese);
			Intent menuIntent = new Intent(ListMenu.this, menuClass);
			startActivity(menuIntent);
		} catch (ClassNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		// TODO Auto-generated method stub
		super.onCreateOptionsMenu(menu);

		//La clase MenuInflater permite crear el menú emergente
		//que surge al pulsar el botón de opciones del dispositivo
		//móvil
		MenuInflater blowUp = getMenuInflater();
		blowUp.inflate(R.menu.sub_menu, menu);
		return true;
	}

	@Override
	public boolean onOptionsItemSelected(MenuItem item) {
		// TODO Auto-generated method stub

		//En el menú emergente se incluyen las opciones de salir de
		//la aplicación (exit) y una breve descripción del proyecto.
		switch (item.getItemId()) {
		case R.id.aboutUs:

			Intent i = new Intent("es.jose.fpapp.ABOUTUS");
			startActivity(i);
			break;

		case R.id.exit:
			finish();
			break;

}
```

```
                return false;
        }

}
```

- **Menú reclutamiento:** *Enrolment.java*

```
public class Enrolment extends Activity implements OnClickListener {

        Button addName, help, clear;
        EditText recruitID;
        TextView recTitle, name;
        ListView recruitList;
        ArrayAdapter<String> myList;
        ArrayList<String> candidateList;
        String listFilePath = "IdList", loadList;
        byte[] listData;
        AlertDialog.Builder adb;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                // TODO Auto-generated method stub
                super.onCreate(savedInstanceState);
                setContentView(R.layout.rmenu);
                initializeRecruitment();
        }
        private void initializeRecruitment() {
                // TODO Auto-generated method stub
                // Inicialización de variables
                addName = (Button) findViewById(R.id.baddName);
                clear = (Button) findViewById(R.id.bClear);
                help = (Button) findViewById(R.id.bHelpR);
                recruitID = (EditText) findViewById(R.id.edID);
                recTitle = (TextView) findViewById(R.id.tvrecruitTitle);
                name = (TextView) findViewById(R.id.tvNames);
                recruitList = (ListView) findViewById(R.id.lvCandidates);

                candidateList = new ArrayList<String>();

                // Instanciación del ArrayAdapter y asignación del mismo a nuestro
                // ListView
                myList = new ArrayAdapter<String>(this,
                                android.R.layout.simple_list_item_1, candidateList);
                recruitList.setAdapter(myList);

                // Asignación de listeners para poder implementar los views cuando se
                // pulsen
                addName.setOnClickListener(this);
                help.setOnClickListener(this);
                clear.setOnClickListener(this);

                // Implementaremos la función del botón de ayuda
                help.setOnClickListener(new View.OnClickListener() {

                        public void onClick(View v) {
                                // TODO Auto-generated method stub
                                Intent i = new Intent("es.jose.fpapp.HELPRECRUIT");
                                startActivity(i);
                        }
                });

                // Si aplicamos una pulsación a uno de los items de la lista, entraremos
                // en la
                // activity para asignar una huella a ese candidato y generar su archivo
                // ".xyt"
                // de minucias. A continuación implementaremos esa funcionalidad
```

```
recruitList.setOnItemClickListener(new OnItemClickListener() {

        public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
                        long arg3) {
                // TODO Auto-generated method stub

                // Adquirimos la información de posición e ID del candidato
                final String itemID = ((TextView) arg1).getText().toString();
                final int itemPos = arg2;

                // Pop-up del alert dialog que nos dará la elección de una
                // detección
                // directa o por pasos
                adb = new AlertDialog.Builder(Enrolment.this);
                adb.setTitle("Steps/Direct");
                adb.setMessage("Direct or step by step detection?");
adb.setPositiveButton("Direct",
                                new AlertDialog.OnClickListener() {
                                        public void onClick(DialogInterface dialog,
                                                        int which) {
                                                Intent goMinDirect = new Intent(Enrolment.this,
                                                                CandidatesMinutia.class);
                                                goMinDirect.putExtra("candidateID", itemID);
                                                goMinDirect.putExtra("candidatePos", itemPos);
                                                try {
                                                        startActivityForResult(goMinDirect, 0);
                                                } catch (Exception e) {
                                                        e.printStackTrace();
                                                        e.getCause();
                                                }
                                        }
                                });

                adb.setNegativeButton("StepByStep",
                                new AlertDialog.OnClickListener() {
                                        public void onClick(DialogInterface dialog,
                                                        int which) {
                                                Intent goMinStep = new Intent(Enrolment.this,
                                                                StepByStep.class);
                                                goMinStep.putExtra("candidateID", itemID);
                                                goMinStep.putExtra("candidatePos", itemPos);
                                                try {
                                                        startActivityForResult(goMinStep, 0);
                                                } catch (Exception e) {
                                                        e.printStackTrace();
                                                        e.getCause();
                                                }

                                        }
                                });
                adb.show();
        }

});

recruitList.setOnItemLongClickListener(new OnItemLongClickListener() {

        // A continuación se implementará la funcionalidad para eliminar un
        // item(candidato)
        // de la lista tras haber realizado una pulsación prolongada sobre
        // el mismo
        public boolean onItemLongClick(AdapterView<?> arg0, View arg1,
                        int arg2, long arg3) {
                // TODO Auto-generated method stub
                final String itemToRemove = ((TextView) arg1).getText()
                                .toString();

                // Pop up de un AlertDialog para asegurar que la pulsación no ha
                // sido por error
                AlertDialog.Builder adb = new AlertDialog.Builder(
                                Enrolment.this);
                adb.setTitle("Remove!");
                adb.setMessage("Are you sure you want to delete "
```

```
                        + itemToRemove);
                        final int positionToRemove = arg2;
                        adb.setNegativeButton("Cancel", null);
                        adb.setPositiveButton("Ok", new AlertDialog.OnClickListener() {
                                public void onClick(DialogInterface dialog, int which) {

                                        // Se borra el item seleccionado y se actualiza la lista
                                        deleteListOrItem(itemToRemove);
                                        candidateList.remove(positionToRemove);
                                        myList.notifyDataSetChanged();

                                        // Posteriormente se procede a guardar la nueva lista
                                        listData = convertArrayListToByteArray(candidateList);
                                        saveNewListOrItem(listData, listFilePath);

                                }
                        });
                        adb.show();

                        return false;
                }

        });

        // Carga de la lista guardada en cada inicio
        loadList = getIdListFile(listFilePath);
        if (loadList != null) {
                byte[] inputCandidates = loadList.getBytes();
                candidateList.addAll(convertByteArrayToArrayList(inputCandidates));
                myList.notifyDataSetChanged();
        } else {
                candidateList.clear();
                myList.notifyDataSetChanged();
        }
}

// Esta funcion se usará para convertir el array de Byte en un ArrayList
// que un bucle rellenara con los candidatos guardados en el fichero
// correspondiente
// y lo devolverá a la lista principal
private Collection<? extends String> convertByteArrayToArrayList(
                byte[] inputCandidates) {
        // TODO Auto-generated method stub
        ArrayList<String> ListaCandidatos = new ArrayList<String>();
        ByteArrayInputStream inByteArray = new ByteArrayInputStream(
                        inputCandidates);
        DataInputStream input = new DataInputStream(inByteArray);
        try {
                while (input.available() > 0) {
                        String candidate = input.readUTF();
                        ListaCandidatos.add(candidate);
                }
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        return ListaCandidatos;
}

// Definimos las acciones que realizará cada view una vez pulsado
public void onClick(View arg0) {
        // TODO Auto-generated method stub
        switch (arg0.getId()) {
        case R.id.baddName:

                // Ocultamos el teclado del teléfono una vez introducido el
                // candidato
            InputMethodManager imm = (InputMethodManager)
                   getSystemService(Context.INPUT_METHOD_SERVICE);
                imm.hideSoftInputFromWindow(recruitID.getWindowToken(), 0);
                String candidateName = recruitID.getText().toString();
                candidateList.add(candidateName);
                myList.notifyDataSetChanged();
```

```
recruitID.setText(null);
                listData = convertArrayListToByteArray(candidateList);
                saveNewListOrItem(listData, listFilePath);

                break;
        case R.id.bClear:

                // Aseguramos con un AlertDialog que el botón de Clear no se pulsó
                // por error
                AlertDialog.Builder adb1 = new AlertDialog.Builder(Enrolment.this);
                adb1.setTitle("Clear!");
                adb1.setMessage("Are you sure you want to clear User's List?");
                adb1.setNegativeButton("Cancel", null);
                adb1.setPositiveButton("Ok", new AlertDialog.OnClickListener() {
                        public void onClick(DialogInterface dialog, int which) {
                                candidateList.clear();
                                myList.notifyDataSetChanged();
                                deleteListOrItem(listFilePath);
                        }
                });
                adb1.show();
                break;
        }

}

private void deleteListOrItem(String listFilePath2) {
        // TODO Auto-generated method stub
        try {
                FileOutputStream removeFile = openFileOutput(listFilePath2,
                                MODE_PRIVATE);
                removeFile.write(null);
                removeFile.close();
        } catch (Exception e) {
                e.printStackTrace();
        }

}

private void saveNewListOrItem(byte[] list, String listFilePath) {
        // TODO Auto-generated method stub
        try {
                FileOutputStream fout = openFileOutput(listFilePath, MODE_PRIVATE);
                fout.write(list);
                fout.close();
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

}

private byte[] convertArrayListToByteArray(ArrayList<String> candidateList2) {
        // TODO Auto-generated method stub
        byte[] updatedList = null;
        ByteArrayOutputStream outputByte = new ByteArrayOutputStream();
        DataOutputStream output = new DataOutputStream(outputByte);
        for (String candidate : candidateList2) {
                try {
                        output.writeUTF(candidate);
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
        updatedList = outputByte.toByteArray();
        return updatedList;
}

public String getIdListFile(String listFilePath2) {
        // TODO Auto-generated method stub
        String getId = null;
        try {
                BufferedReader inputIdList = new BufferedReader(
```

```
                        new InputStreamReader(openFileInput(listFilePath2)));
                        getId = inputIdList.readLine().toString();
                        inputIdList.close();
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                return getId;
        }

}
```

- **Activity detección:** *CandidatesMinutia.java*

```
public class CandidatesMinutia extends Activity {

        Button load, detect, save, help;
        TextView pth1, user, title;
        ImageView input;

        String userID, userXytPath, filePath;
        int userPos;
        Intent gallery = new Intent();
        private static final int requestImg = 1;
        Bitmap bmp;
        long tStart, tEnd, tOffset;

        minutiae[] minutiaSave = new minutiae[1];

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                // TODO Auto-generated method stub
                super.onCreate(savedInstanceState);
                setContentView(R.layout.detect);
                initializeVars();
                collectItemData();
                user.setTextColor(Color.MAGENTA);

                // Implementamos la función del botón de ayuda
                help.setOnClickListener(new View.OnClickListener() {

                        public void onClick(View v) {
                                // TODO Auto-generated method stub
                                Intent i = new Intent("es.jose.fpapp.HELPDETECT");
                                startActivity(i);
                        }
                });

                // Implementación del browser para elegir la imagen (desde el botón
                // "load")
                load.setOnClickListener(new View.OnClickListener() {

                        public void onClick(View v) {
                                // TODO Auto-generated method stub
                                gallery.setType("image/*");
                                gallery.setAction(Intent.ACTION_GET_CONTENT);
                                startActivityForResult(
                                                Intent.createChooser(gallery, "Select Image"),
                                                requestImg);
                        }
                });

                // Implementación del código que generará el archivo de minucias ".xyt"
                detect.setOnClickListener(new View.OnClickListener() {

                        public void onClick(View v) {
                                // TODO Auto-generated method stub
```

```
SE PROCEDE A LA DETECCIÓN DE LAS MINUCIAS
                        tEnd = System.nanoTime();
                        tOffset = tEnd - tStart;
                        long tOffsetMillis = tOffset / 1000000;
                        String time = "Detection time: "
                                        + String.valueOf(tOffsetMillis);
                        pth1.setText(time + " ms");
                        pth1.setTextColor(Color.RED);
                        minutiaSave[0] = ominutiae[0];
                        input.setImageBitmap(bmp);
                        Context context = getApplicationContext();
                        CharSequence text = ".xyt file generated";
                        int duration = Toast.LENGTH_SHORT;
                        Toast toastMin = Toast.makeText(context, text, duration);
                        toastMin.show();

                } else {
                        try {
                                CharSequence noImg = "Load an image first";
                                Toast noImage = Toast.makeText(CandidatesMinutia.this,
                                                noImg, Toast.LENGTH_LONG);
                                noImage.show();
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }

        }

});
save.setOnClickListener(new View.OnClickListener() {

        public void onClick(View v) {
                // TODO Auto-generated method stub
                // Comprobamos que existe una imagen cargada

                if (filePath != null) {

                        // Comprobamos que existe archivo de minucias
                        if (minutiaSave[0] != null) {

                                try {
                                        // A continuación ejecutamos el método encargado de
                                        // guardar los resultados, para ello almacenaremos
                                        // el
                                        // fichero en la memoria interna del teléfono con el
                                        // siguiente
                                        // OutputStreamWriter lo cual dará una mayor
                                        // seguridad a
                                        // la hora
                                        // de proteger los datos de los usuarios

                                        OutputStreamWriter fout = new OutputStreamWriter(
                                                        openFileOutput(userXytPath,
                                                                        Context.MODE_PRIVATE));
                                        saveXytFile(minutiaSave, fout);
                                        CharSequence tx = "XYT file added to " + userID;
                                        int dur = Toast.LENGTH_SHORT;
                                        Toast toast = Toast.makeText(
                                                        CandidatesMinutia.this, tx, dur);
                                        toast.show();
                                        CandidatesMinutia.this.finish();

                                } catch (Exception ex) {
                                        ex.printStackTrace();
                                        CharSequence text1 = "Error in saving .xyt file";
                                        int duration1 = Toast.LENGTH_SHORT;
                                        Toast toast = Toast.makeText(
                                                        CandidatesMinutia.this, text1, duration1);
                                        toast.show();
                                }

                        } else
```

```
                        try {

                                        CharSequence noMin = "Detect minutiae first";
                                        Toast noXyt = Toast.makeText(
                                                        CandidatesMinutia.this, noMin,
                                                        Toast.LENGTH_LONG);
                                        noXyt.show();
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                        } else
                                try {

                                        CharSequence noMin = "Load an image first";
                                        Toast noXyt = Toast.makeText(CandidatesMinutia.this,
                                                        noMin, Toast.LENGTH_LONG);
                                        noXyt.show();
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                }

        });
}
private void collectItemData() {
        // TODO Auto-generated method stub

        // Recogida de datos
        Bundle getData = getIntent().getExtras();
        userID = getData.getString("candidateID");
        userPos = getData.getInt("candidatePos");
        user.setText("User: " + userID);

        // Creamos el path donde se guardará el archivo ".xyt" del sujeto
        // en cuestión
        // File rutaSD = Environment.getExternalStorageDirectory();
        // String savingRoute = rutaSD.getAbsolutePath() + "/Android/";
        // String interPath = savingRoute + userID;
        userXytPath = userID + ".xyt";

}

private void initializeVars() {
        // TODO Auto-generated method stub

        load = (Button) findViewById(R.id.bLoad);
        detect = (Button) findViewById(R.id.bDetect);
        save = (Button) findViewById(R.id.bSave);
        help = (Button) findViewById(R.id.bHelp);
        pth1 = (TextView) findViewById(R.id.tvPth1);
        input = (ImageView) findViewById(R.id.ivLoad);
        user = (TextView) findViewById(R.id.tvUser);
        title = (TextView) findViewById(R.id.tvDetTitle);
}

// Función que servirá para guardar el archivo de minucias asociado al
// usuario
// seleccionado
private void saveXytFile(minutiae[] ominutiae, OutputStreamWriter fout) {
        // TODO Auto-generated method stub
        int i, oq;
        int[] ox, oy, ot;
        ox = new int[1];
        oy = new int[1];
        ot = new int[1];
        minutia[] minutia = new minutia[1];
        for (i = 0; i < ominutiae[0].num; i++) {
                minutia[0] = ominutiae[0].list[i];
                xytreps.lfs2m1_minutia_XYT(ox, oy, ot, minutia[0]);
                // guardamos cada minucia.
                oq = lfs_objects.sround(minutia[0].reliability * 100.0);
                int mox = ox[0];
                int moy = oy[0];
```

```
        int mot = ot[0];
                        String s = String.format("%d %d %d %d", mox, moy, mot, oq);

                        try {
                                fout.write(s + "\r\n");
                        } catch (Exception e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
                try {
                        fout.close();
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        }

        @Override
        protected void onActivityResult(int requestCode, int resultCode, Intent data) {
                // TODO Auto-generated method stub
                super.onActivityResult(requestCode, resultCode, data);
                if (requestCode == requestImg) {

                        Uri selectedImgUri = data.getData();
                        String[] filePathColumn = { MediaStore.Images.Media.DATA };

                        Cursor cursor = getContentResolver().query(selectedImgUri,
                                        filePathColumn, null, null, null);
                        cursor.moveToFirst();

                        int columnIndex = cursor.getColumnIndex(filePathColumn[0]);
                        filePath = cursor.getString(columnIndex);
                        cursor.close();
                        bmp = BitmapFactory.decodeFile(filePath);
                        input.setImageBitmap(bmp);

                }

        }

}
```

- **Activity detección por pasos:** *StepByStep.java*

```
public class StepByStep extends Activity {

        Button loadSbutton, step, saveS, helpS;
        TextView pth1S, userS, titleS, processTimer;
        ImageView inputS;

        String userIDS, userXytPathS, filePathS;
        int userPosS;
        Intent galleryS = new Intent();
        private static final int requestImgS = 1;
        Bitmap bmpS, overlay;
        long tStartS, tEndS, tOffsetS;
        int contador = 0;
        Canvas canvas;
        Paint paint;

        minutiae[] minutiaSave = new minutiae[1];

        // Strings que contendrán la breve descripción de cada proceso

        String binarDescription = "BINARIZATION: in this point the input fingerprint image has "
                        + "been binarized. White pixels for valleys and black for ridges.";
```

```
String detectDescription = "DETECTION: in this point the detected minutia are "
                + "shown within the original fingerprint image.";

@Override
protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.steps);
        initializeVars();
        collectItemData();
        userS.setTextColor(Color.MAGENTA);

        // Implementamos la función del botón de ayuda
        helpS.setOnClickListener(new View.OnClickListener() {

                public void onClick(View v) {
                        // TODO Auto-generated method stub
                        Intent i = new Intent("es.jose.fpapp.HELPDETECT");
                        startActivity(i);
                }
        });

        // Implementación del browser para elegir la imagen (desde el botón
        // "load")
        loadSbutton.setOnClickListener(new View.OnClickListener() {

                public void onClick(View v) {
                        // TODO Auto-generated method stub
                        galleryS.setType("image/*");
                        galleryS.setAction(Intent.ACTION_GET_CONTENT);
                        startActivityForResult(
                                        Intent.createChooser(galleryS, "Select Image"),
                                        requestImgS);
                }
        });

        step.setOnClickListener(new View.OnClickListener() {

                public void onClick(View v) {
                        // TODO Auto-generated method stub

                        if (filePathS != null) {
                                switch (contador) {

                                case 0:


                                        SE PROCEDE A LA DETECCIÓN DE LAS MINUCIAS
                                        // Fin de la cuenta de tiempo
                                        tEndS = System.nanoTime();
                                        tOffsetS = tEndS - tStartS;
                                        bz_gbls.tOverallTime = tOffsetS / 1000000;

                                        minutiaSave[0] = ominutiae[0];


                                        //Comenzamos la implementación de los círculos rojos que
                                        //rodearán
                                        //las minucias en la imagen original. Para ello se crea un
                                        //Bitmap
                                        //que se utilizará como una capa en la que se pintarán estos //círculos
                                        //a través de la función drawCircle()
                                        BitmapFactory.Options bfo = new BitmapFactory.Options();
                                        bfo.inSampleSize = 5;
                                        overlay = Bitmap.createBitmap(mapa.getWidth(),
                                                        mapa.getHeight(), mapa.getConfig());
                                        canvas = new Canvas(overlay);
                                        paint = new Paint();
                                        paint.setColor(Color.RED);
                                        paint.setStyle(Style.STROKE);
                                        paint.setStrokeWidth((float) 2.5);
                                        canvas.drawBitmap(mapa, new Matrix(), null);
```

```
                //Se recorren los centros de cada una de las minucias
                //para rodearlos mediante circulos rojos, cuyo color, estilo
                //y ancho
                //se han especificado anteriormente.
                for (int d = 0; d < minutiaSave[0].num; d++) {
                        int m_x = minutiaSave[0].list[d].x;
                        int m_y = minutiaSave[0].list[d].y;
                        canvas.drawCircle(m_x, m_y, 10, paint);
                }

                inputS.setImageBitmap(bz_gbls.binar);
                contador++;

                // Procedemos a rellenar los campos que contendrán la
                // información de tiempo
                // y descripción del proceso

                pth1S.setText(binarDescription);
                pth1S.setTextColor(Color.WHITE);

                processTimer.setText("Binarization time = "
                                + bz_gbls.tBinarMillis + " ms");
                processTimer.setTextColor(Color.GREEN);

                break;

        case 1:

                inputS.setImageBitmap(overlay);
                Context context = getApplicationContext();
                CharSequence text = ".xyt file generated";
                int duration = Toast.LENGTH_SHORT;
                Toast toastMin = Toast
                                .makeText(context, text, duration);
                toastMin.show();
                step.setText("Finish");
                contador++;

                // Procedemos a rellenar los campos que contendrán la
                // información de tiempo
                // y descripción del proceso

                pth1S.setText(detectDescription);
                pth1S.setTextColor(Color.WHITE);

                processTimer.setText("Detection time = "
                                + bz_gbls.tDetectMillis + " ms");
                processTimer.setTextColor(Color.GREEN);

                break;

        case 2:

                pth1S.setText("OVERALL PROCESS");
                pth1S.setTextColor(Color.WHITE);

                processTimer.setText("Overall time = " + bz_gbls.tOverallTime
                                + " ms");
                processTimer.setTextColor(Color.GREEN);

                step.setText("Exit without save");
                contador++;

                break;

        case 3:

                finish();

                break;

        }
```

```
} else {
                        try {
                                CharSequence noImg = "Load an image first";
                                Toast noImage = Toast.makeText(StepByStep.this, noImg,
                                                Toast.LENGTH_LONG);
                                noImage.show();
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }
        }

});

saveS.setOnClickListener(new View.OnClickListener() {

        public void onClick(View v) {
                // TODO Auto-generated method stub
                // Comprobamos que existe una imagen cargada

                if (filePathS != null) {

                        // Comprobamos que existe archivo de minucias
                        if (minutiaSave[0] != null) {

                                try {
                                        // A continuación ejecutamos el método encargado de
                                        // guardar los resultados, para ello almacenaremos
                                        // el
                                        // fichero en la memoria interna del teléfono con el
                                        // siguiente
                                        // OutputStreamWriter lo cual dará una mayor
                                        // seguridad a
                                        // la hora
                                        // de proteger los datos de los usuarios

                                        OutputStreamWriter fout = new OutputStreamWriter(
                                                        openFileOutput(userXytPathS,
                                                                        Context.MODE_PRIVATE));
                                        saveXytFile(minutiaSave, fout);
                                        CharSequence tx = "XYT file added to " + userIDS;
                                        int dur = Toast.LENGTH_SHORT;
                                        Toast toast = Toast.makeText(StepByStep.this, tx,
                                                        dur);
                                        toast.show();
                                        StepByStep.this.finish();

                                } catch (Exception ex) {
                                        ex.printStackTrace();
                                        CharSequence text1 = "Error in saving .xyt file";
                                        int duration1 = Toast.LENGTH_SHORT;
                                        Toast toast = Toast.makeText(StepByStep.this,
                                                        text1, duration1);
                                        toast.show();
                                }

                        } else
                                try {

                                        CharSequence noMin = "Detect minutiae first";
                                        Toast noXyt = Toast.makeText(StepByStep.this,
                                                        noMin, Toast.LENGTH_LONG);
                                        noXyt.show();
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                } else
                        try {

                                CharSequence noMin = "Load an image first";
                                Toast noXyt = Toast.makeText(StepByStep.this, noMin,
                                                Toast.LENGTH_LONG);
                                noXyt.show();
```

```
                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                }

        });
}

private void collectItemData() {
        // TODO Auto-generated method stub

        // Recogida de datos
        Bundle getData = getIntent().getExtras();
        userIDS = getData.getString("candidateID");
        userPosS = getData.getInt("candidatePos");
        userS.setText("User: " + userIDS);

        // Creamos el path donde se guardará el archivo ".xyt" del sujeto
        // en cuestión
        // File rutaSD = Environment.getExternalStorageDirectory();
        // String savingRoute = rutaSD.getAbsolutePath() + "/Android/";
        // String interPath = savingRoute + userID;
        userXytPathS = userIDS + ".xyt";

}

private void initializeVars() {
        // TODO Auto-generated method stub

        loadSbutton = (Button) findViewById(R.id.bLoadgoStep);
        step = (Button) findViewById(R.id.bGoDetectStep);
        saveS = (Button) findViewById(R.id.bSaveStep);
        helpS = (Button) findViewById(R.id.bHelpStep);
        pth1S = (TextView) findViewById(R.id.tvPth1Step);
        inputS = (ImageView) findViewById(R.id.ivLoadStep);
        userS = (TextView) findViewById(R.id.tvUserStep);
        titleS = (TextView) findViewById(R.id.tvStepTitle);
        processTimer = (TextView) findViewById(R.id.tvTimerSteps);
        processTimer.setTextColor(Color.RED);
}

// Función que servirá para guardar el archivo de minucias asociado al
// usuario
// seleccionado
private void saveXytFile(minutiae[] ominutiae, OutputStreamWriter fout) {
        // TODO Auto-generated method stub
        int i, oq;
        int[] ox, oy, ot;
        ox = new int[1];
        oy = new int[1];
        ot = new int[1];
        minutia[] minutia = new minutia[1];
        for (i = 0; i < ominutiae[0].num; i++) {
                minutia[0] = ominutiae[0].list[i];
                xytreps.lfs2m1_minutia_XYT(ox, oy, ot, minutia[0]);
                // guardamos cada minucia.
                oq = lfs_objects.sround(minutia[0].reliability * 100.0);
                int mox = ox[0];
                int moy = oy[0];
                int mot = ot[0];
                String s = String.format("%d %d %d %d", mox, moy, mot, oq);

                try {
                        fout.write(s + "\r\n");
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }
        try {
                fout.close();
        } catch (Exception e) {
                // TODO Auto-generated catch block
```

```
		e.printStackTrace();
			}

		}

		@Override
		protected void onActivityResult(int requestCode, int resultCode, Intent data) {
			// TODO Auto-generated method stub
			super.onActivityResult(requestCode, resultCode, data);
			if (requestCode == requestImgS) {
				Uri selectedImgUri = data.getData();
				String[] filePathColumn = { MediaStore.Images.Media.DATA };

				Cursor cursor = getContentResolver().query(selectedImgUri,
						filePathColumn, null, null, null);
				cursor.moveToFirst();

				int columnIndex = cursor.getColumnIndex(filePathColumn[0]);
				filePathS = cursor.getString(columnIndex);
				cursor.close();
				bmpS = BitmapFactory.decodeFile(filePathS);
				inputS.setImageBitmap(bmpS);
			}

		}

}
```

- **Menú verficación:** *Verification.java*

```
public class Verification extends Activity implements OnClickListener {

		TextView title;
		ListView verifList;
		ArrayAdapter<String> myAdapter;
		ArrayList<String> usersList;
		String loadList, listFilePath = "IdList";

		@Override
		protected void onCreate(Bundle savedInstanceState) {
			// TODO Auto-generated method stub
			super.onCreate(savedInstanceState);
			setContentView(R.layout.verifylist);
			initialize();

			// Carga de la lista de candidatos rellenada en la Activity recruitMenu
			loadList = getIdListFile(listFilePath);
			if (loadList != null) {
				byte[] inputCandidates = loadList.getBytes();
				usersList.addAll(convertByteArrayToArrayList(inputCandidates));
				myAdapter.notifyDataSetChanged();
			} else {
				usersList.clear();
				myAdapter.notifyDataSetChanged();
			}

		}

		private Collection<? extends String> convertByteArrayToArrayList(
				byte[] inputCandidates) {
			// TODO Auto-generated method stub
			ArrayList<String> ListaCandidatos = new ArrayList<String>();
			ByteArrayInputStream inByteArray = new ByteArrayInputStream(
					inputCandidates);
			DataInputStream input = new DataInputStream(inByteArray);
			try {
				while (input.available() > 0) {
```

```
                String candidate = input.readUTF();
                                ListaCandidatos.add(candidate);
                        }
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                return ListaCandidatos;
        }

        private String getIdListFile(String listFilePath2) {
                // TODO Auto-generated method stub
                String getId = null;
                try {
                        BufferedReader inputIdList = new BufferedReader(
                                        new InputStreamReader(openFileInput(listFilePath2)));
                        getId = inputIdList.readLine().toString();
                        inputIdList.close();
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                return getId;
        }

        private void initialize() {
                // TODO Auto-generated method stub
                title = (TextView) findViewById(R.id.tvVerifTitle);
                verifList = (ListView) findViewById(R.id.lvVerif);

                usersList = new ArrayList<String>();
                myAdapter = new ArrayAdapter<String>(this,
                                android.R.layout.simple_list_item_1, usersList);
                verifList.setAdapter(myAdapter);

                // Implementacion de la selección de dos candidatos para la comparación
                verifList.setOnItemClickListener(new OnItemClickListener() {

                        public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
                                        long arg3) {
                                // TODO Auto-generated method stub

                                final String itemID = ((TextView) arg1).getText().toString();
                                final int itemPos = arg2;
                                Intent goVerif = new Intent(Verification.this,
                                                VerifyActivity.class);
                                goVerif.putExtra("candidateID", itemID);
                                goVerif.putExtra("candidatePos", itemPos);
                                startActivityForResult(goVerif, 0);

                        }

                });

        }

        public void onClick(View v) {
                // TODO Auto-generated method stub

        }

}
```

- **Activity verificación:** *VerifyActivity.java*

```
public class VerifyActivity extends Activity {

	TextView verifInput, verifResult, user, titleVerif;
	Button verify, verifImage, help;
	ImageView vImage;
	BufferedReader inputFile, userFile;

	String userID, filePath, resultNotif;
	int userPos   , auxResult;
	Intent browser = new Intent();
	private static final int requestImg = 1;
	Bitmap bmp;
	String huellaInputPath;
	long timeEnd, timeStart, offset;

	// Creamos la variable de tipo minucia que contendrá el archivo ".xyt" que
	// resulte
	// de la huella de entrada
	minutiae[] minutiaSave = new minutiae[1];

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		// TODO Auto-generated method stub
		super.onCreate(savedInstanceState);
		setContentView(R.layout.verifyactivity);
		initialize();
		collectItemData();

		// Implementamos la función del botón de ayuda
		help.setOnClickListener(new View.OnClickListener() {

			public void onClick(View v) {
				// TODO Auto-generated method stub
				Intent i = new Intent("es.jose.fpapp.HELPVERIF");
				startActivity(i);
			}
		});

		// La pusación de este View provocará la comparación entre el ".xyt"
		// autogenerado
		// de la huella de entrada y el ".xyt" del usuario seleccionado
		verify.setOnClickListener(new View.OnClickListener() {

			public void onClick(View v) {
				// TODO Auto-generated method stub

				if (filePath != null) {
					// Autogeneración del ".xyt" entrante
					String imagePath = filePath;
					autogenerateInputXYT(imagePath);

					// Comenzamos a contar el tiempo de proceso
					timeStart = System.nanoTime();

					// Obtenemos la ruta donde se encuentra el archivo ".xyt"
					// del usuario en cuestión
					String rutaUser = userID + ".xyt";
					String rutaHuella = huellaInputPath;
					if (rutaUser != null && !rutaUser.equals(" ")
							&& rutaHuella != null && !rutaHuella.equals("")) {

						try {
							BufferedReader finput = new BufferedReader(
									new InputStreamReader(
											openFileInput(rutaHuella)));
							inputFile = finput;
							BufferedReader fuser = new BufferedReader(
									new InputStreamReader(
											openFileInput(rutaUser)));
							userFile = fuser;
						} catch (FileNotFoundException e) {
							// TODO Auto-generated catch block
							e.printStackTrace();
```

```
                        }
                                xyt_struct user = bz_io.bz_load(userFile);
                                xyt_struct fingerPrint = bz_io.bz_load(inputFile);
                                int RESULT = bz_drvrs.bozorth_main(user, fingerPrint);
                                auxResult = RESULT;
                                if (RESULT > 250) {
                                        resultNotif = "Accepted";
                                        Bitmap accepted = BitmapFactory.decodeResource(
                                                        getResources(), R.drawable.granted);
                                        vImage.setImageBitmap(accepted);

                                } else {
                                        resultNotif = "Denied";
                                        Bitmap deniedBmp = BitmapFactory.decodeResource(
                                                        getResources(), R.drawable.denied);
                                        vImage.setImageBitmap(deniedBmp);
                                }
                                verifResult.setText("Verification result :"
                                                + resultNotif);
                                bz_gbls.reset();
                        } else {
                                verifResult
                                                .setText("Formato de datos de entrada
                                                                incorrectos");
                        }

                        // Finalizamos la cuenta del tiempo
                        timeEnd = System.nanoTime();
                        offset = timeEnd - timeStart;
                        long offsetMillis = offset / 1000000;
                        String time = "Verification time: "
                                        + String.valueOf(offsetMillis);
                        verifInput.setText(time + " ms");
                        verifInput.setTextColor(Color.RED);

                } else {
                        try {
                                CharSequence noV = "Load an image first";
                                Toast noVerif = Toast.makeText(VerifyActivity.this,
                                                noV, Toast.LENGTH_LONG);
                                noVerif.show();
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }
                String auxText = Integer.toString(auxResult);
                user.setText(auxText);
        }
    });

    // Implementación del browser para elegir la imagen (desde el botón
    // "load")
    verifImage.setOnClickListener(new View.OnClickListener() {

        public void onClick(View v) {
                // TODO Auto-generated method stub
                browser.setType("image/*");
                browser.setAction(Intent.ACTION_GET_CONTENT);
                startActivityForResult(
                                Intent.createChooser(browser, "Select Image"),
                                requestImg);
        }
    });
}

private void initialize() {
    // TODO Auto-generated method stub
    titleVerif = (TextView) findViewById(R.id.tvtituloVer);

    verifImage = (Button) findViewById(R.id.bLoadV);
    verifInput = (TextView) findViewById(R.id.tvPthV);
    user = (TextView) findViewById(R.id.tvUserV);
    verifResult = (TextView) findViewById(R.id.tvVerifResult);
```

```
        verify = (Button) findViewById(R.id.bVerifV);
        help = (Button) findViewById(R.id.bHelpV);
        vImage = (ImageView) findViewById(R.id.ivLoadV);
}

private void collectItemData() {
        // TODO Auto-generated method stub

        // Recogida de datos
        Bundle getData = getIntent().getExtras();
        userID = getData.getString("candidateID");
        userPos = getData.getInt("candidatePos");
        user.setText("User: " + userID);
        user.setTextColor(Color.MAGENTA);

}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        // TODO Auto-generated method stub
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == requestImg) {
                Uri selectedImgUri = data.getData();
                String[] filePathColumn = { MediaStore.Images.Media.DATA };

                Cursor cursor = getContentResolver().query(selectedImgUri,
                                filePathColumn, null, null, null);
                cursor.moveToFirst();

                int columnIndex = cursor.getColumnIndex(filePathColumn[0]);
                filePath = cursor.getString(columnIndex);
                cursor.close();
                bmp = BitmapFactory.decodeFile(filePath);
                vImage.setImageBitmap(bmp);
        }

}

private void saveXytFile(minutiae[] ominutiae, OutputStreamWriter fout) {
        // TODO Auto-generated method stub
        int i, oq;
        int[] ox, oy, ot;
        ox = new int[1];
        oy = new int[1];
        ot = new int[1];
        minutia[] minutia = new minutia[1];
        for (i = 0; i < ominutiae[0].num; i++) {
                minutia[0] = ominutiae[0].list[i];
                xytreps.lfs2m1_minutia_XYT(ox, oy, ot, minutia[0]);
                // guardamos cada minucia.
                oq = lfs_objects.sround(minutia[0].reliability * 100.0);
                int mox = ox[0];
                int moy = oy[0];
                int mot = ot[0];
                String s = String.format("%d %d %d %d", mox, moy, mot, oq);

                try {
                        fout.write(s + "\r\n");
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }
        try {
                fout.close();
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

}

private void autogenerateInputXYT(String imagePath) {
```

```
                android.graphics.ImageFormat[] imgfrm = new android.graphics.ImageFormat[1];
                short[] data = new short[1];
                int[] ow = new int[1], oh = new int[1], od = new int[1], oppi = new int[1];
                ow[0] = -1;
                oh[0] = -1;
                minutiae min = new minutiae(1000);
                minutiae[] ominutiae = new minutiae[1];
                ominutiae[0] = min;
                int[] oqualitymap, olow_c, olowf, odir_m, ohigh;

                short[] obdata;
                int[] omapw = new int[1], omaph = new int[1], obw = new int[1],
                        obh = new int[1], obd = new int[1];
                omapw[0] = -1;
                omaph[0] = -1;
                obw[0] = -1;
                obh[0] = -1;
                // Decofidicamos imagen en escala de grises
                data = imgdecod.read_and_decode_grayscale_image(imagePath, imgfrm,
                                data, ow, oh, od, oppi, vImage);

                LFSPARMS lf = new LFSPARMS();
                lf.setValuesV2();
                // Inicialización de obdata
                int bdataSize = ow[0] * oh[0];
                obdata = new short[bdataSize];
                // Inicialización de los mapas
                int width = (int) Math.ceil(ow[0] / (double) LFSPARMS.blocksize);
                int heigth = (int) Math.ceil(oh[0] / (double) LFSPARMS.blocksize);
                int mapSize = width * heigth;
                oqualitymap = new int[mapSize];
                olow_c = new int[mapSize];
                olowf = new int[mapSize];
                odir_m = new int[mapSize];
                ohigh = new int[mapSize];
                // try {
                // Procedemos a obtener el array de minucias y sus
                // caraterísticas
                getmin.get_minutiae(ominutiae, oqualitymap, odir_m, olow_c, olowf,
                                ohigh, omapw, omaph, obdata, obw, obh, obd, data, ow[0], oh[0],
                                8, 500 / 25.4, lf, vImage);

                minutiaSave[0] = ominutiae[0];
                if (minutiaSave[0].num != 0) {
                        // Creamos el fichero que contendrá nuestro archivo xyt de
                        // salida
                        String interPath = "minutiaeInput";
                        String inputMinPath = interPath + ".xyt";
                        huellaInputPath = inputMinPath;
                        try {
                                // A continuación ejecutamos el método encargado de
                                // guardar los resultados

                                OutputStreamWriter fout = new OutputStreamWriter(
                                                openFileOutput(huellaInputPath, Context.MODE_PRIVATE));

                                // DataOutputStream fout = new DataOutputStream(
                                // openFileOutput(userXytPath,
                                // Context.MODE_PRIVATE));
                                saveXytFile(minutiaSave, fout);

                        } catch (Exception ex) {
                                ex.printStackTrace();
                                CharSequence text1 = "Error in saving input minutia .xyt file";
                                int duration1 = Toast.LENGTH_SHORT;
                                Toast toast = Toast.makeText(VerifyActivity.this, text1,
                                                duration1);
                                toast.show();
                        }

                }

        }
}
```

- **Activity identificación:** *Identification.java*

```
public class Identification extends Activity {

	Button help, load, identify;
	ImageView imageID;
	TextView IDResult, titleIden;
	String listFilePath = "IdList", filePath, huellaInputPath, user,
			userIdentified, printFile;
	ArrayList<String> usersList = new ArrayList<String>();
	minutiae[] minutiaSave = new minutiae[1];
	Intent browserID = new Intent();
	ArrayAdapter<String> userAdapter;
	private static final int requestImg = 1;
	Bitmap bmp;
	int cont = 0;
	int cont2 = 0;
	int resultado = 0;
	BufferedReader printReader, userReader;
	AlertDialog.Builder adb;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		// TODO Auto-generated method stub
		super.onCreate(savedInstanceState);
		setContentView(R.layout.identification);
		initialize();

		// Recorremos la lista para comparar e identificar a quién
		// corresponde
		// la huella

		identify.setOnClickListener(new View.OnClickListener() {

			public void onClick(View v) {

				if (filePath != null) {

					int i = 0;
					int RESULT = 0;

					// Cargamos la lista de usuarios

					String loadList = getIdListFile(listFilePath);
					byte[] inputCandidates = loadList.getBytes();
					usersList.removeAll(usersList);
					userAdapter.notifyDataSetChanged();

					usersList
							.addAll(convertByteArrayToArrayList(inputCandidates));
					userAdapter.notifyDataSetChanged();
					// Reiniciamos el string que contendrá nuestro usuario
					// válido
					// y aumentamos el contador que nos dirá el número de veces
					// que
					// se
					// ha pulsado el botón de indentificar
					if (cont == 0) {
						// TODO Auto-generated method stub
						IDResult.setTextColor(Color.WHITE);
						IDResult.setText("Working...");

						while ((RESULT < 25) & (i < usersList.size())) {

							user = usersList.get(i);
							String userFile = user + ".xyt";
```

```
                try {
                        BufferedReader finput = new BufferedReader(
                                        new InputStreamReader(
                                                openFileInput(printFile)));
                        printReader = finput;
                        BufferedReader fuser = new BufferedReader(
                                        new InputStreamReader(

                        openFileInput(userFile)));
                        userReader = fuser;
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                xyt_struct usuario = bz_io.bz_load(userReader);
                xyt_struct huella = bz_io.bz_load(printReader);
                RESULT = bz_drvrs.bozorth_main(usuario, huella);
                i++;

        }

        // Aquí hemos abandonado el bucle, por lo que ha
        // encontrado al
        // usuario o no hay ninguno con esa huella

        // Si lo ha encontrado...

        if (RESULT > 25) {

                try {
                        i = 0;
                        RESULT = 0;
                        usersList.removeAll(usersList);
                        userAdapter.notifyDataSetChanged();
                        userIdentified = user;
                        IDResult.setText("User accpted: "
                                        + userIdentified);
                        IDResult.setTextColor(Color.YELLOW);
                        user = null;
                        userIdentified = null;
                        CharSequence noID = "Identification completed";
                        Toast noIdent = Toast.makeText(
                                        Identification.this, noID,
                                        Toast.LENGTH_LONG);
                        noIdent.show();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        // Si no...
        else {
                try {
                        i = 0;
                        RESULT = 0;
                        usersList.removeAll(usersList);
                        userAdapter.notifyDataSetChanged();
                        user = null;
                        userIdentified = null;
                        IDResult.setText("Null ID");
                        CharSequence noID = "No user with this
                                                        fingerprint";
                        Toast noIdent = Toast.makeText(
                                        Identification.this, noID,
                                        Toast.LENGTH_LONG);
                        noIdent.show();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        // Si el botón de identificar se pulsa una segunda vez
        // se
```

```
                        // abrirá
                                // un AlertDialog que nos dará dos opciones: la de
                                // realizar
                                // una nueva
                                // identificación (reset de activity), o la de salir
                                // finalizando la
                                // activity.

                        } else {

                                showAlertDialog(adb);

                        }
                        cont++;

                } else {
                        try {
                                CharSequence noV = "Load an image first";
                                Toast noVerif = Toast.makeText(Identification.this,
                                                noV, Toast.LENGTH_LONG);
                                noVerif.show();
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }
        }

});

// Implementamos la función del botón de ayuda
help.setOnClickListener(new View.OnClickListener() {

        public void onClick(View v) {
                // TODO Auto-generated method stub
                Intent i = new Intent("es.jose.fpapp.HELPIDENTIFY");
                startActivity(i);
        }
});

// Implementación del browser para elegir la imagen (desde el botón
// "load")
load.setOnClickListener(new View.OnClickListener() {

        public void onClick(View v) {

                if (cont2 == 0) {
                        // TODO Auto-generated method stub
                        cont2++;
                        IDResult.setTextColor(Color.WHITE);
                        IDResult.setText("Loading image...");
                        browserID.setType("image/*");
                        browserID.setAction(Intent.ACTION_GET_CONTENT);
                        startActivityForResult(
                                        Intent.createChooser(browserID, "Select Image"),
                                        requestImg);
                } else {

                        showAlertDialog(adb);
                }
        }
});

}

private void autogenerateInputXYT(String imagePath) {
        // TODO Auto-generated method stub
        SE REALIZA LA DETECCIÓN DE LAS MINUCIAS DE LA HUELLA DE ENTRADA

        minutiaSave[0] = ominutiae[0];
        if (minutiaSave[0].num != 0) {
                // Creamos el fichero que contendrá nuestro archivo xyt de
                // salida
                String interPath = "minutiaeInput";
```

```
String inputMinPath = interPath + ".xyt";
                huellaInputPath = inputMinPath;
                printFile = huellaInputPath;
                try {
                        // A continuación ejecutamos el método encargado de
                        // guardar los resultados

                        OutputStreamWriter fout = new OutputStreamWriter(
                                        openFileOutput(huellaInputPath, Context.MODE_PRIVATE));

                        saveXytFile(minutiaSave, fout);

                } catch (Exception ex) {
                        ex.printStackTrace();
                        CharSequence text1 = "Error in saving input minutia .xyt file";
                        int duration1 = Toast.LENGTH_SHORT;
                        Toast toast = Toast.makeText(Identification.this, text1,
                                        duration1);
                        toast.show();
                }

        }

}

private void saveXytFile(minutiae[] minutiaSave2, OutputStreamWriter fout) {
        // TODO Auto-generated method stub
        int i, oq;
        int[] ox, oy, ot;
        ox = new int[1];
        oy = new int[1];
        ot = new int[1];
        minutia[] minutia = new minutia[1];
        for (i = 0; i < minutiaSave2[0].num; i++) {
                minutia[0] = minutiaSave2[0].list[i];
                xytreps.lfs2m1_minutia_XYT(ox, oy, ot, minutia[0]);
                // guardamos cada minucia.
                oq = lfs_objects.sround(minutia[0].reliability * 100.0);
                int mox = ox[0];
                int moy = oy[0];
                int mot = ot[0];
                String s = String.format("%d %d %d %d", mox, moy, mot, oq);

                try {
                        fout.write(s + "\r\n");
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }
        try {
                fout.close();
        } catch  (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

}

private Collection<? extends String> convertByteArrayToArrayList(
                byte[]inputCandidates){
```

```
        // TODO Auto-generated method stub
        ArrayList<String> ListaCandidatos = new ArrayList<String>();
        ByteArrayInputStream inByteArray = new ByteArrayInputStream(
                        inputCandidates);
        DataInputStream input = new DataInputStream(inByteArray);
        try {
                while (input.available() > 0) {
                        String candidate = input.readUTF();
                        ListaCandidatos.add(candidate);
                }
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        return ListaCandidatos;
}

private String getIdListFile(String listFilePath2) {
        // TODO Auto-generated method stub
        String getId = null;
        try {
                BufferedReader inputIdList = new BufferedReader(
                                new InputStreamReader(openFileInput(listFilePath2)));
                getId = inputIdList.readLine().toString();
                inputIdList.close();
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        return getId;
}

private void initialize() {
        // TODO Auto-generated method stub
        help = (Button) findViewById(R.id.bHelpI);
        load = (Button) findViewById(R.id.bLoadI);
        identify = (Button) findViewById(R.id.bIdentify);
        imageID = (ImageView) findViewById(R.id.ivLoadI);
        IDResult = (TextView) findViewById(R.id.tvIDResult);
        titleIden = (TextView) findViewById(R.id.tvTitleIden);
        userAdapter = new ArrayAdapter<String>(this,
                        android.R.layout.simple_list_item_1, usersList);
        cont = 0;

}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        // TODO Auto-generated method stub
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == requestImg) {
                Uri selectedImgUri = data.getData();
                String[] filePathColumn = { MediaStore.Images.Media.DATA };

                Cursor cursor = getContentResolver().query(selectedImgUri,
                                filePathColumn, null, null, null);
                cursor.moveToFirst();

                int columnIndex = cursor.getColumnIndex(filePathColumn[0]);
                filePath = cursor.getString(columnIndex);
                cursor.close();
                bmp = BitmapFactory.decodeFile(filePath);
                // Autogeneración del ".xyt" entrante
                autogenerateInputXYT(filePath);
                imageID.setImageBitmap(bmp);
                IDResult.setText("Image load");

        }
}
```

```
	private void showAlertDialog(Builder adb) {
		// TODO Auto-generated method stub
		adb = new AlertDialog.Builder(Identification.this);
		adb.setTitle("Reset/exit");
		adb.setMessage("New identification or exit?");

		adb.setPositiveButton("New", new AlertDialog.OnClickListener() {
			public void onClick(DialogInterface dialog, int which) {
				Intent reset = new Intent(Identification.this,
						Identification.class);
				Identification.this.finish();
				startActivityForResult(reset, 0);
			}
		});
		adb.setNegativeButton("Exit", new AlertDialog.OnClickListener() {
			public void onClick(DialogInterface dialog, int which) {
				Identification.this.finish();

			}
		});
		adb.show();
	}
}
```

- **Splash:** *Splash.java*

```
public class Splash extends Activity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		// TODO Auto-generated method stub
		super.onCreate(savedInstanceState);
		setContentView(R.layout.main);
		Thread timer = new Thread() {
			public void run() {
				try {
					sleep(2000);
					finish();
				} catch (InterruptedException e) {
					e.printStackTrace();
				} finally {
					Intent start = new Intent("es.jose.fpapp.LISTMENU");
					startActivity(start);
				}
			}

};
		timer.start();
	}

}
```